Imperial College London

ELEC70001 - Adaptive Signal Processing and Machine Intelligence
Coursework Assignment

Author:
Sakshi Singh - 01925165

Date:
April 2, 2024

Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

The Power Spectral Density (PSD) of a signal is a spectral representation of its power content at different frequencies. The PSD has two common definitions:

Definition 1: The PSD is defined as the DTFT of the Autocovariance function (ACF).

$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k)e^{-j\omega k} \tag{1}$$

where $r(k)$ is the autocovariance function of the signal $x(n)$.

Definition 2: The PSD is derived from the time-limited signal's Fourier Transform squared and averaged over time.

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

1.1.1 Analytical Proof of Equivalence of the 2 Definitions of PSD

The autocovariance function for a wide-sense stationary (WSS) process or a random signal $x(n)$ is defined as:

$$r(k) = E[x(n) \cdot x^*(n-k)] \tag{3}$$

The ACF reflects how signal values $x(n)$ are related to their values at other times $x(n+k)$.
Taking the DTFT of $r(k)$ directly gives us the first definition of PSD in (1).
The Fourier Transform of the signal, squared and normalized, leads to the second definition of PSD. Expanding the squared magnitude:

$$\frac{1}{N} \left| \sum_{n=0}^{N-1} x(n)e^{-j\omega n} \right|^2 = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(n)x^*(m)e^{-j\omega(n-m)} \tag{4}$$

Given $x(n)x^*(m) = E[x(n)x^*(m)] = r(n-m)$, we substitute to obtain:

$$\frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} r(n-m)e^{-j\omega(n-m)} \tag{5}$$

Given that $r(k) = r(-k)$ for real signals, and considering $n - m = k$, the expression for the Power Spectral Density (PSD) in (5) simplifies to the Discrete-Time Fourier Transform (DTFT) of $r(k)$, thereby matching Definition 1.

The key assumption for the equivalence is that the autocovariance sequence $r(k)$ decays rapidly:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k||r(k)| = 0 \tag{6}$$

This assumption guarantees that long-range correlations (large $|k|$) do not significantly contribute to the overall power, allowing the signal's PSD to be successfully captured by both the autocovariance function and the direct Fourier analysis.

Moreover this equivalence is a direct consequence of the Fourier transform's linearity and symmetry properties, combined with the stationarity assumption that allows $r(k)$ to fully represent the signal's power across frequencies.

The detailed mathematical transition is represented as follows:

$$\frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(n)x^*(m)e^{-j\omega(n-m)} = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} r(n-m)e^{-j\omega(n-m)} \tag{7}$$

Given $r(k) = r(-k)$ for real signals and considering the change of variable $n-m = k$, we can rewrite the summation in terms of k, leading to the DTFT of the autocovariance function $r(k)$:

$$\frac{1}{N} \sum_{k=-(N-1)}^{N-1} \sum_{n=0}^{N-1} r(k)e^{-j\omega k} \tag{8}$$

As N approaches infinity, and under the mild assumption that the autocorrelation sequence $r(k)$ decays rapidly enough, this expression converges to the DTFT of $r(k)$, which is:

$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k)e^{-j\omega k} \tag{9}$$

1.1.2 Simulation showing Scenario where Equivalence exists

To show equivalence, we can consider a simple signal with a rapidly decaying autocorrelation function such as a white noise signal, which has an autocovariance function that is a delta (δ) function (i.e., it decays instantly).

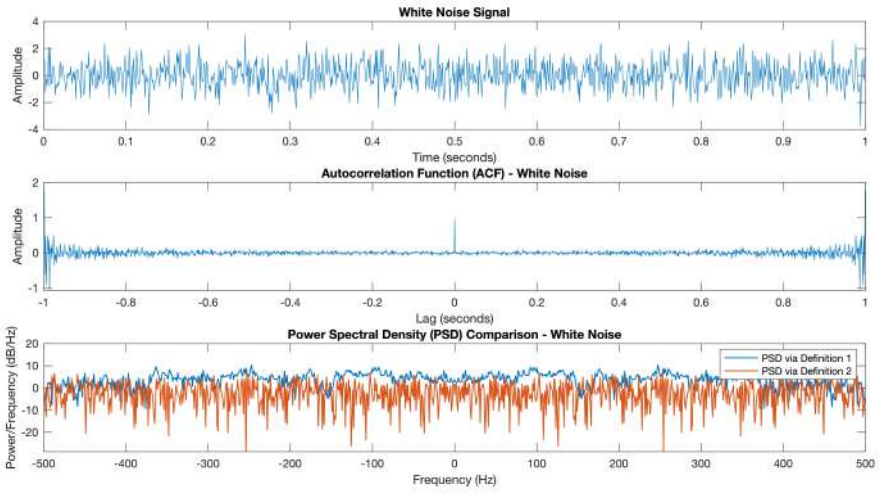

Figure 1: White noise signal, its ACF and PSD computed using both definitions. Definition 1 refers to Eq. (1) and Definition 2 refers to Eq. (2)

Fig.1 shows that the PSD computed using the 2 definitions are similar as they overlap, illustrating the equivalence under the condition of rapid decay in the autocovariance sequence.

1.1.3 Simulation showing scenario where Equivalence does not exist

For the non-equivalence scenario, we consider a signal with a slow-decaying autocovariance function, such as a low-frequency sinusoid or a signal with long-term dependencies.

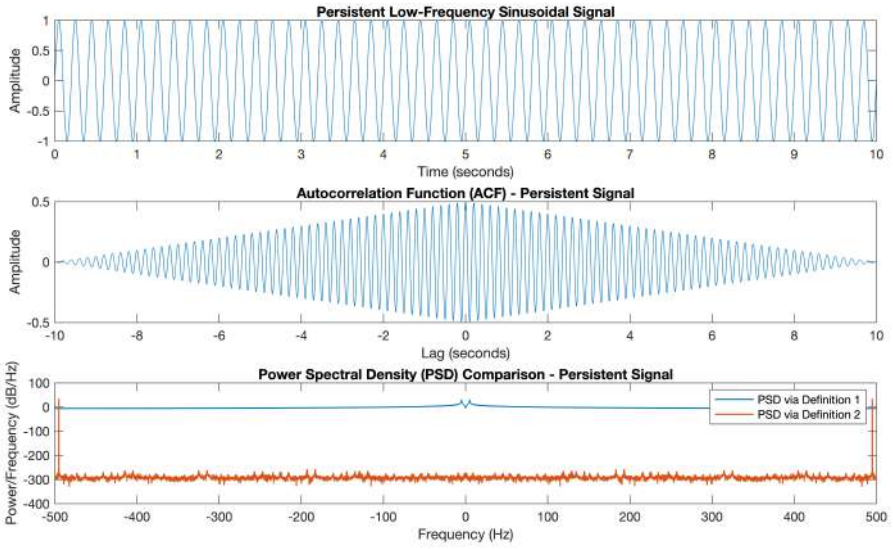

Figure 2: Persistent Low Frequency Sinusoidal signal, its ACF and PSD computed using both definitions. Definition 1 refers to Eq. (1) and Definition 2 refers to Eq. (2)

Fig.2 shows that the PSD computed are different, illustrating that the equivalence does not hold as the assumption of rapid decay of autocorrelation sequence is violated in this scenario.

1.2 Periodogram-based Methods Applied to Real–World Data

1.2.1 (a) Periodogram-based Methods on Sunspot Time Series

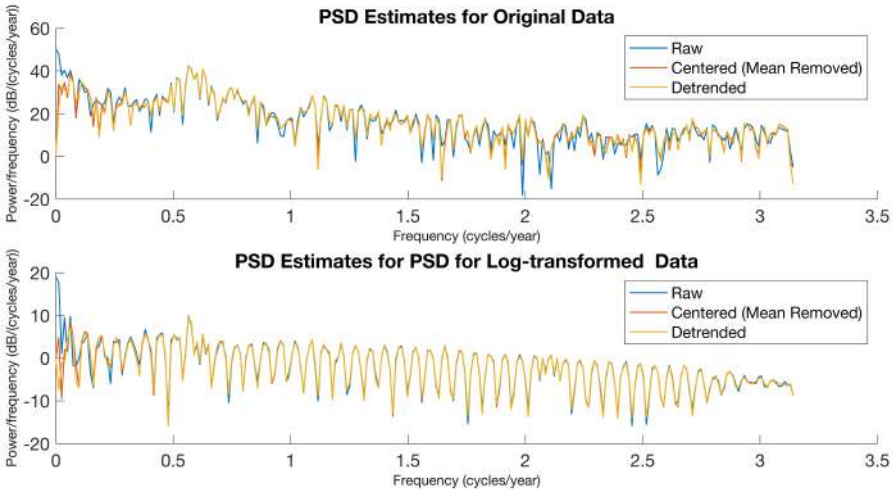

Figure 3: PSD Estimates for original sunspot data (Top) and Log-transformed data (Bottom)

The mean removal from original sunspot data in Fig.3 (Top) effectively centers the data around zero. By doing so the average sunspot count over the years is subtracted from each data point. It removes the DC (zero-frequency) component from the spectral estimate.
The trend removal eliminates any linear trend or variations in the data. This helps in capturing the cyclic and periodic components of the data without interference from long-term trends.
As a result, periodicities that were obscured by the mean and trend become more prominent in the PSD estimate of the original sunspot data. After applying a logarithmic transformation to the sunspot data as seen in Fig.3 (Bottom), each data sample is replaced with its logarithm.
This effectively compresses the data, emphasizing smaller variations while reducing the impact of large values. Taking the logarithm can also stabilize variance, making the data more suitable for analysis.
As a result, logarithmic transformation can emphasize periodicities that are present in the data on a relative scale, allowing smaller variations in sunspot counts to become more apparent.

1.2.2 (b) Periodogram-based Methods on EEG Signals - The Basis for Brain Computer Interface (BCI).

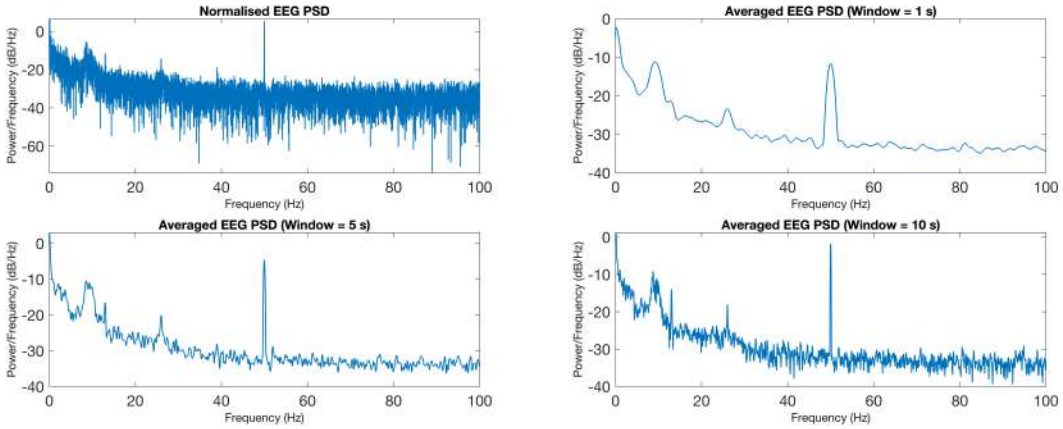

Figure 4: Normalised PSD estimates of EEG Signal for standard periodogram and averaged periodogram of different window lengths

From the 'Normalised EEG PSD' in Fig. 4 we can observe 4 smaller peaks at around 8.5 Hz, 13 Hz, 26 hHz, 39 hz and a large peak at around 50 Hz. The peak at 50 Hz is due to the power-line interference that was induced in the

recording apparatus. The peak at 8.5 Hz is due to alpha-rhythm of the subject being tested. The peak at 13 Hz corresponds to the SSVEP and is the fundamental frequency response peak. The peaks at 26 and 39 Hz are the the harmonics of the SSVEP peak.

Comparing Standard Periodogram with Averaged Periodogram of Window = 10 s

In spectral analysis of EEG signals, the standard periodogram is typically characterized by high variance because it relies on a single estimate from the entire signal duration, leading to potential fluctuations that can obscure true spectral content.
On the other hand, an averaged periodogram using a 10-second window length inherently possesses lower variance due to the averaging of multiple periodograms from overlapped signal segments. This averaging process smooths out random noise, making genuine spectral peaks such as SSVEP more discernible against other EEG activity.
However, this decrease in variance is at the expense of increased bias, notably spectral smoothing, which can blur closely spaced frequency components together. This increase in bias is seen in 'Averaged EEG PSD (Window = 10s)' in Fig.4 where the harmonic at 39 Hz is not easily distinguishable compared to standard periodogram.

Effect of making the Window Size Small

When employing an averaged periodogram approach with a very small window size, such as 1 second, the primary effect is a reduction in frequency resolution due to the decreased width of the analysis window.
This can make it difficult to distinguish between closely spaced frequency components, as they may be merged into a single peak in the power spectrum.
This is seen 'Averaged EEG PSD (Window = 1s)' in Fig.4 where the alpha-rhythm peak and fundamental SSVEP peak at 13 Hz are hard to differentiate.
Additionally, while shorter windows do decrease the variance of the spectral estimate by averaging across more segments, each segment contains less information about the signal, which can result in a less accurate representation of the true spectral content.

1.3 Correlation Estimation

1.3.1 (a) Unbiased Correlation Estimation and Preservation of Non-negative Spectra

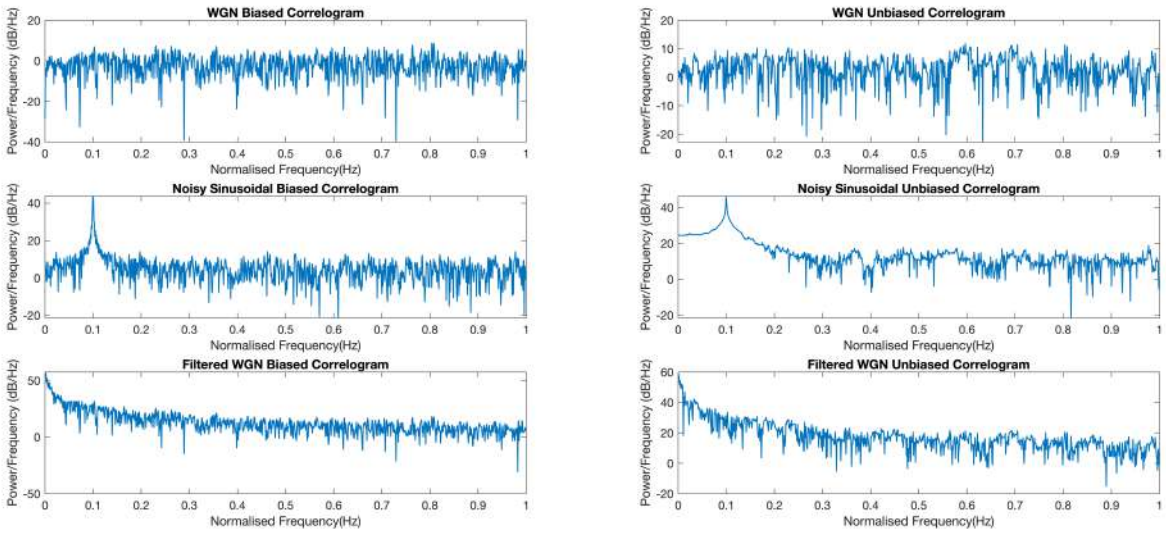

Figure 5: Correlogram for biased (Left) and unbiased (Right) WGN, noisy sinusoidal signals and filtered WGN signals

A MATLAB script was created to calculate the correlogram based on both biased and unbiased ACF calculations. The script was then validated using White Gaussian Noise (WGN), noisy sinusoidal signals, and filtered WGN as seen in Fig.5

Spectral Estimates Difference

The biased ACF estimate tends to smooth out the spectral estimate, as it gives equal weight to all data points, potentially leading to an overestimation of spectral power. In contrast, the unbiased estimator weights the terms based on the number of data pairs available, aiming to represent the true autocorrelation more accurately. However, this

can result in higher variance in the spectral estimate. This can be seen in the plot in Fig. 5 for each type of signal.

Unbiased Correlogram Behavior at Large Lags (k)

The correlogram corresponding to the unbiased ACF estimates can be highly erratic for large lags, where the number of data pairs used in the calculation diminishes. This sparsity of data can lead to an unreliable estimation of the autocorrelation function, resulting in a noisy and unstable correlogram, especially at higher frequencies where the contribution of the signal is limited.

Negative Values in Estimated PSD

The unbiased ACF estimate can result in negative values for the estimated PSD. This occurs because the unbiased estimate can lead to an autocorrelation sequence that is not positive definite, particularly when estimated from a noisy signal or at large lags. Since the PSD is essentially the Fourier transform of the autocorrelation sequence, any non-positive-definite behavior in the time domain can manifest as negative values in the frequency domain, which are physically not interpretable.

1.3.2 (b) & (c) Plotting the PSD in dB

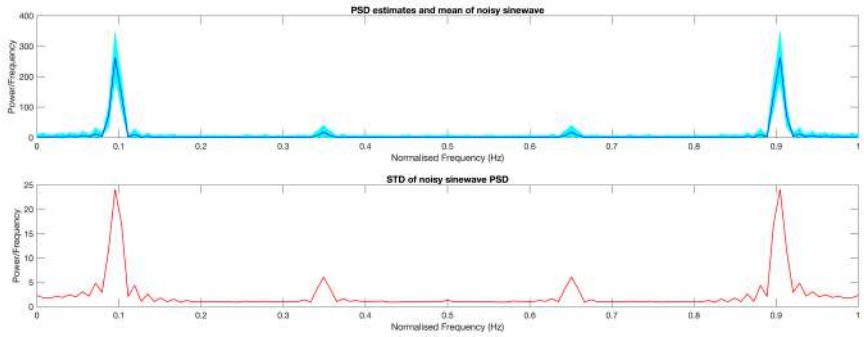

Figure 6: PSD Estimate of 100 realisations of noisy sinusoidal signal plotted in the linear scale with its standard deviation.

(b) PSD Estimates of Multiple Realizations

When generating the Power Spectral Density (PSD) estimates of several realizations of a random process, we can observe varying degrees of dispersion among the realizations as seen in Fig. 6.
Each realization of the noisy signal will have a different noise pattern, leading to slight differences in the estimated PSD.
The spectral peaks corresponding to the sinusoids' frequencies tend to be consistent across realizations, but the noise generates a varying background level.
When plotted, the realizations show a clustering around the sinusoid frequencies with their power levels, while the noise creates a 'fuzzy' appearance around the baseline.

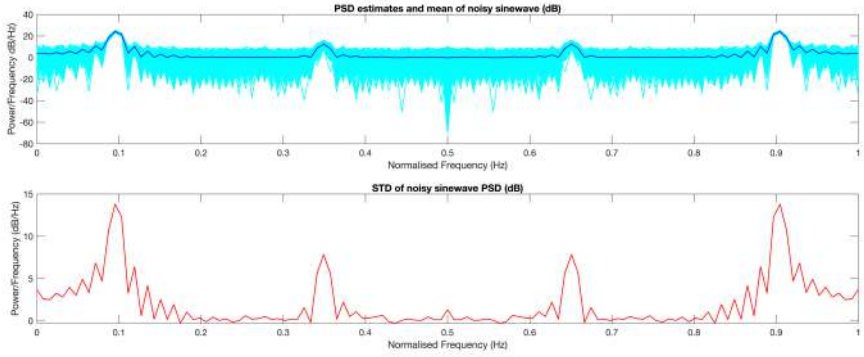

Figure 7: PSD Estimate of 100 realisations of noisy sinusoidal signal plotted in the dB scale with its standard deviation.

(c) PSD Estimates in Decibel Scale

Plotting the PSD estimates in decibels (dB) along with their associated standard deviation provides a more visually condensed representation of the dispersion among the realizations as seen in Fig. 7.
The logarithmic nature of the decibel scale compresses the wide dynamic range of the power levels, making it easier to interpret the spread of the estimates. The peaks corresponding to the sinusoids are still distinguishable, but the variability due to noise is less visually pronounced when compared to the linear scale.

1.3.3 (d) & (e) Frequency Estimation by MUSIC.

(d) Effect of Sample Length on Periodogram of Complex Exponential Signals

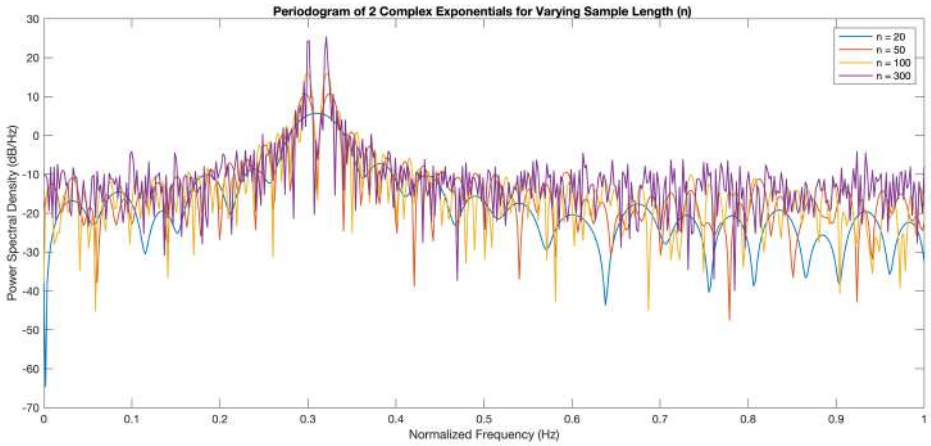

Figure 8: Periodogram of two complex exponentials with closely-spaced frequencies for different values of Sample Length.

The frequency resolution of the periodogram can be approximated by $1/N$, where N is the length of the time-domain signal or the number of samples used in the Fourier Transform.
The frequency resolution determines the smallest frequency difference that can be distinguished by the periodogram. If two frequencies are closer together than this resolution, the periodogram will not be able to resolve them as two separate spectral peaks. Instead, it may show them as a single peak or not accurately represent their true frequencies.

For instance, with two complex exponentials at frequencies of 0.3 Hz and 0.32 Hz, the frequency separation between them is 0.02 Hz. To resolve these two frequencies using a periodogram, the frequency resolution needs to be finer than this separation.

For N=20 samples, the frequency resolution is 0.05 Hz, which is greater than the separation of 0.02 Hz between the two frequencies. Thus, the periodogram cannot resolve these two frequencies at this sample size.

Increasing N to 50, 100, or 300 samples improves the frequency resolution to 0.02 Hz, 0.01 Hz, and 0.0033 Hz, respectively. With these finer resolutions, especially at N=100 and N=300, the periodogram starts to show distinct peaks for the two frequencies because the resolution is equal to or finer than the separation between the frequencies.

(e) Finding the Desired Line Spectra using the MUSIC Method

MUSIC (Multiple Signal Classification) algorithm, which is an advanced signal processing technique used for estimating the frequencies of sinusoidal waves within a signal, particularly in the presence of noise. This algorithm is part of the subspace-based methods, which are different from traditional Fourier transform methods like the periodogram. The MUSIC algorithm operates by assuming that a noisy signal can be modeled by a sum of complex exponentials, and it leverages the eigendecomposition of the signal's autocorrelation matrix. The key insight of MUSIC is that it separates the signal into two orthogonal subspaces: the signal subspace, spanned by the eigenvectors corresponding to the largest eigenvalues, and the noise subspace, spanned by the eigenvectors corresponding to the smallest eigenvalues. The frequencies of the signal components are estimated by finding the peaks in the MUSIC spectrum, which are identified where the signal's steering vector is orthogonal to the noise subspace.

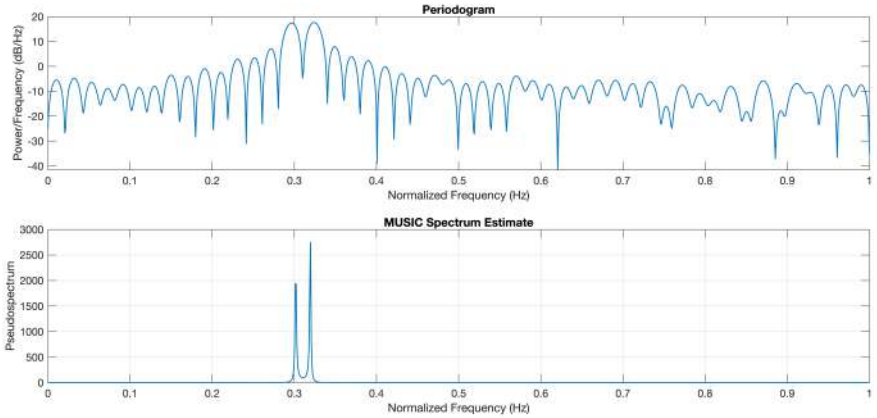

Figure 9: Periodogram and MUSIC Pseudospectrum of noisy complex exponential signal with two frequency components

Code Explanation

The function `corrmtx(x,14,'mod')` generates a correlation matrix **X** from the input signal **x** using the modified covariance method. The second argument, 14, specifies the order of the matrix or the number of lags to be used in the correlation matrix calculation. The string '`mod`' indicates that the modified covariance method is to be used.

Operation: This function is used to estimate the autocorrelation matrix of the input signal **x**. The autocorrelation matrix is fundamental in subspace methods like MUSIC, as it captures the statistical properties of the signal that are utilized to distinguish between signal and noise components.

The command `pmusic(R,2,[],1,'corr')` applies the MUSIC algorithm to the correlation matrix **R** obtained from `corrmtx`. The second argument, 2, specifies the number of sinusoidal components in the signal. The third argument is empty `[]`, indicating that the default number of FFT points, which is 256, is used for the pseudospectrum computation. The fourth argument 1 specifies the sampling frequency, and '`corr`' indicates that the input matrix **R** is a correlation matrix.

Operation: This function computes the MUSIC pseudospectrum **S** over the frequency range specified by **F**. The MUSIC algorithm operates by identifying the subspace spanned by the eigenvectors of the signal's correlation matrix associated with its largest eigenvalues (signal subspace) and distinguishing it from the subspace associated with noise.

The line `plot(F,S,'linewidth',2);` sets the line width to 2 and plots the MUSIC pseudospectrum **S** against the frequency **F**. The command `set(gca,'xlim',[0.25 0.40]);` limits the x-axis (frequency axis) to the range 0.25 Hz to 0.40 Hz, focusing on the area of interest.

Operation: Visualizes the estimated spectrum, allowing for the identification of the frequencies of the sinusoidal components in the signal, even if they are closely spaced.

Comparison of Periodogram and MUSIC

	Periodogram	**MUSIC**
Advantages	Simple to implement; effective for broad-spectrum analysis.	High resolution capable of distinguishing closely spaced frequencies; robust to noise in identifying signal components.
Disadvantages	Limited resolution; susceptibility to leakage and variance issues.	Requires prior knowledge of the number of sinusoids; computationally intensive; performance can degrade in low SNR.
Bias and Variance	Generally biased due to leakage and windowing effects. High variance without averaging.	Considered unbiased with lower variance for frequency estimates, but dependent on correct model order selection.

Table 1: Comparison of Periodogram and MUSIC Algorithms

Accuracy of MUSIC Alogrithm

The accuracy of the MUSIC algorithm in estimating the spectrum is generally high for signals that match its model (a sum of sinusoids in white noise), especially when the number of samples is sufficient to accurately estimate the autocorrelation matrix and when the number of sources (sinusoids) is correctly specified. It excels in resolving closely spaced frequencies that the periodogram cannot, making it highly effective for applications requiring precise frequency estimation.

1.4 Spectrum of Autoregressive Processes

1.4.1 (a) Shortcomings of unbiased ACF to estimate AR parameters

The unbiased estimator of the ACF, as given in Eq. (13) in the coursework document, divides by N-k instead of N as in the case of the biased estimator. While this aims to provide an estimate that's unbiased in expectation, it can result in high variance, especially for larger lags (k close to N).This is because, as k increases, the number of terms in the summation decreases, leading to a less reliable estimate due to less averaging.
As a result, high variance can affect the stability and accuracy of the estimated AR parameters.

We perform $\mathbf{a} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xx}$, which relies on being able to invert the autocorrelation matrix to estimate the AR coefficients. An optimal ACF matrix for AR parameter estimation should be Toeplitz and positive definite, which ensures AR system stability.
However, using the unbiased estimate, a smaller number of samples with larger lags may result in an estimated ACF matrix that is not positive-definite. This can result in estimated AR parameters that fail to accurately represent a stable system.

1.4.2 (b) Estimating the PSD of the Signal with Varying AR Model Orders (p)

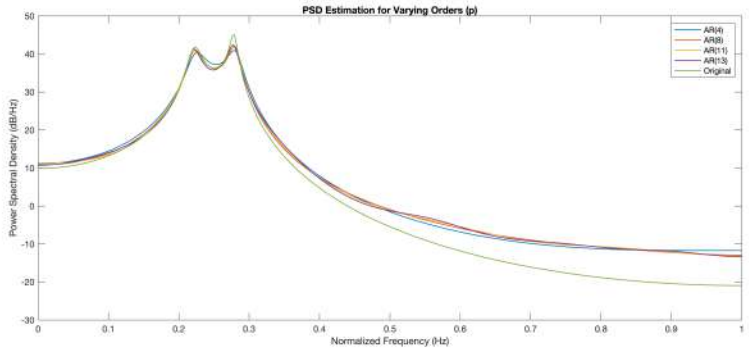

Figure 10: Estimated PSD of 1000 data samples for varying model orders.

As you increase the order 'p' of the AR model, you allow the model to capture more complexity in the data. For a low model order, the AR process may not be able to capture the true dynamics of the signal. This was seen when AR(2) was plotted on MATLAB.
As the model order increases, the estimated PSD should converge towards the true PSD of the signal as the model order becomes sufficient to capture the signal's dynamics. This can be seen in Fig. 10 as the plot of AR(13) fits the true PSD more closely than plot of AR(4).

1.4.3 (c) Estimating the PSD of the Signal with Varying AR Model Orders for 10,000 data samples

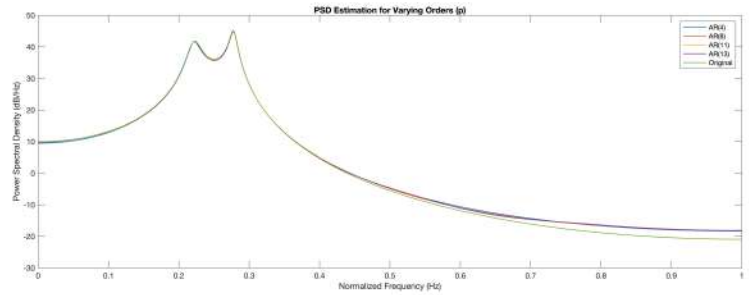

Figure 11: Estimated PSD of 10,000 data samples for varying model orders.

Fig. 11 shows the estimated PSD for different AR models like in Fig. 10 but with 10,000 data samples.
If the model order is too low (under-modelling), we may not capture all the dynamics of the true system, leading to a poor fit and inaccurate PSD estimation. Peaks in the PSD that correspond to the true system frequencies may not be represented, or their magnitudes might be underestimated.

Conversely, if the model order is too high (over-modelling), we start to fit the noise in the data, which introduces spurious peaks in the PSD that do not correspond to any real frequencies of the system.

For an AR(4) process, the correct model order is 4, and using this order should provide the most accurate PSD estimation, closely matching the true PSD of the system. If the number of samples is large, such as 10,000, the PSD estimation becomes more reliable, and the effects of over-modelling and under-modelling can be more clearly observed due to the increased data set size.

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

1.5.1 (a) Plot of the Standard Periodogram and Averaged Periodogram of RRI Data from 3 Trials

To study the effect of Respiratory Sinus Arrhythmia using real data measured using an ECG under different breathing conditions. ECG Data from the Statistical Signal Processing module from Year 3 were used in this analysis to produce the PSD in Fig. 12, 13 and 14.

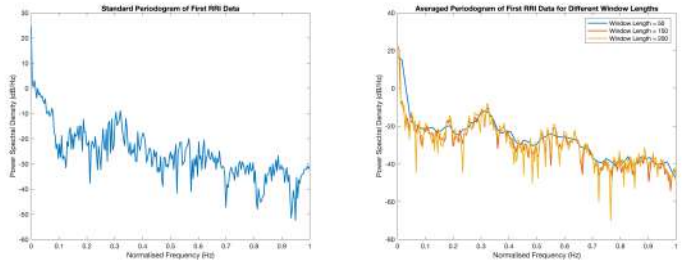

Figure 12: Estimated Standard (Left) and Averaged PSD (Right) from Trial 1 Data

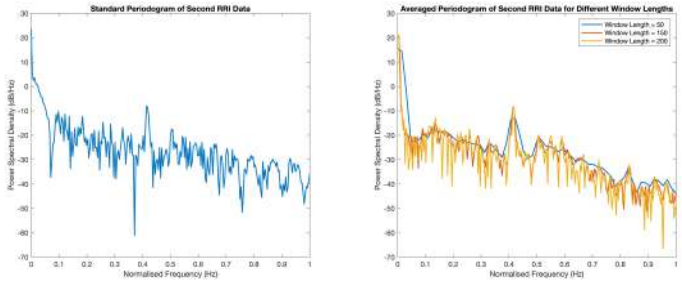

Figure 13: Estimated Standard (Left) and Averaged PSD (Right) from Trial 2 Data

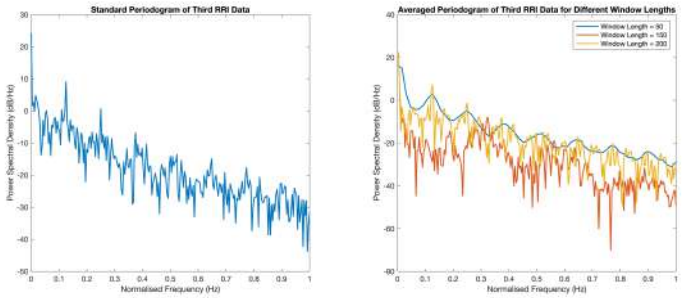

Figure 14: Estimated Standard (Left) and Averaged PSD (Right) from Trial 3 Data

1.5.2 (b) Differences between the PSD Estimates of the RRI Data from the Three Trials

The differences in PSD estimates from the three trials can be attributed to variations in the heart rate due to different breathing patterns, physical conditions, or experimental settings.

Changes in respiration rate can lead to variations in the strength of RSA or peak amplitude, which is reflected in the PSD estimates.
The peaks in the spectrum corresponding to frequencies of respiration can usually be identified within the typical breathing frequency range (about 0.1 to 0.4 Hz for adults at rest). In Trial 1 a peak is seen at around 0.33 Hz, in Trial 2 at around 0.4 Hz and in Trial 3 at around 0.13 Hz. Harmonic peaks also appear at integer multiples of the respiratory frequency.

1.5.3 (c) AR Spectrum Estimate for the RRI Signals

AR spectrum estimation is done on the RRI data from the three trials, and the spectrum is plotted in Fig. 15,16 and 17.

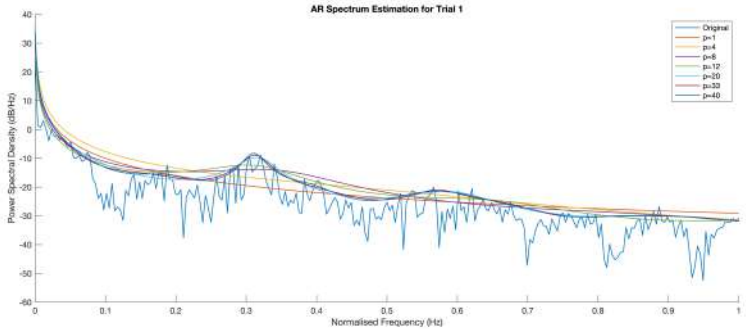

Figure 15: Estimated AR spectrum for Trial 1

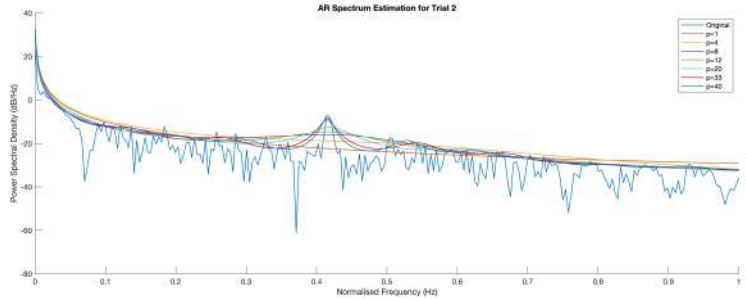

Figure 16: Estimated AR spectrum for Trial 2

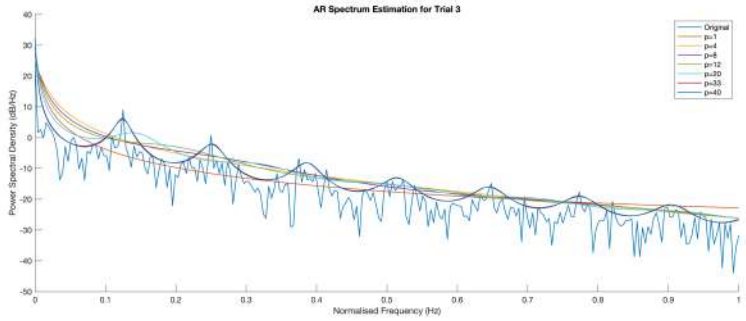

Figure 17: Estimated AR spectrum for Trial 3

Overall from all the three figures above it can be seen that the AR model can offer higher frequency resolution than the periodogram.
The AR spectrum shows a clearer peak corresponding to the respiration rate due to the model's ability to more accurately represent the signal's underlying structure.

The standard periodogram for each trail is more sensitive to noise in the data. It exhibits peaks at various frequencies, including those related to noise, which may not have physiological significance.

1.6 Robust Regression

1.6.1 (a) Identifying the Rank of X and $\mathbf{X}_{noise}$

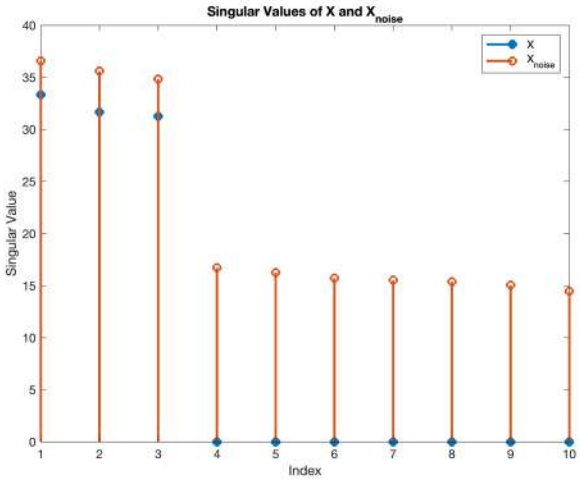

Figure 18: Eigenvalues of X

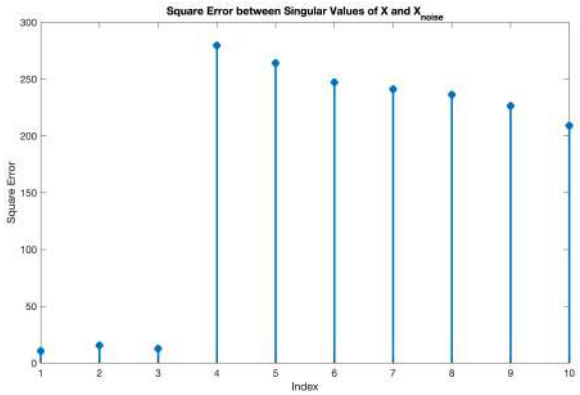

Figure 19: Eigenvalues of X_{noise}

From Fig.18, we observe that the singular values of X (blue) and X_{noise} (orange) start off similarly for the first few indices, which suggests that the leading singular values (which correspond to the most significant components of the data) are relatively unaffected by the noise. However, as we move to the higher indices, the singular values of X_{noise} increase compared to those of X.

The rank of the matrix X can typically be assessed by the number of non-negligible singular values, which is 3 as seen in Fig.18.
In the presence of noise, determining this rank becomes challenging.

The second plot illustrates the square error between the singular values of X and X_{noise}. The low square error for the first few singular values reinforces that the primary components of the data are robust against noise.
Conversely, the square errors for the latter singular values are much larger, highlighting the disproportionate impact of noise on these less significant components. This pattern of error could be attributed to the variance introduced by noise.

1.6.2 (b) Low-rank Approximation of $\mathbf{X}_{noise}$

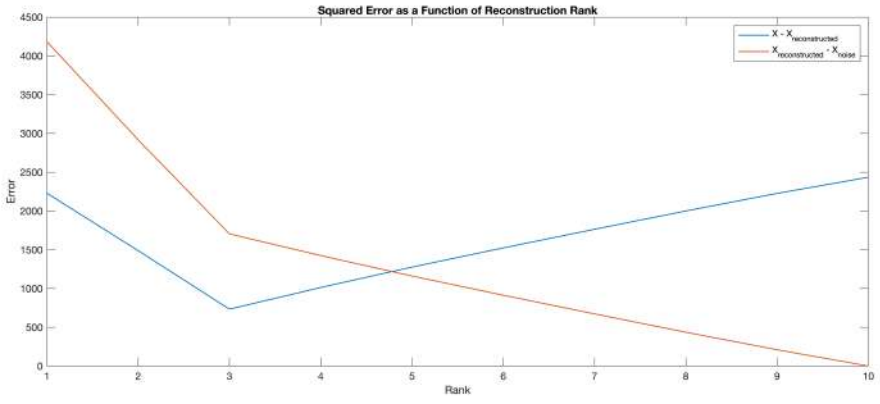

Figure 20: Squared Error as a Function of Reconstruction Rank

Given the observed decrease in reconstruction error at rank 3 for both $X - \tilde{X}_{\text{noise}}$ where $\tilde{X}_{\text{noise}}$ is $\mathrm{X}_{reconstructed}$ in the plot in Fig.20 and $\tilde{X}_{\text{noise}} - X_{\text{noise}}$, we can infer that the first three principal components capture a substantial amount of the significant structure of the noiseless input matrix X.
This rank (3) signifies a balance point where the addition of further components does not significantly improve the representation of X, as the principal components beyond this point may primarily capture noise rather than useful signal. Consequently, the denoised matrix $\tilde{X}_{\text{noise}}$ reconstructed with these three components effectively reduces the noise while retaining the essential characteristics of X, which is evidenced by the minimized errors at this rank.

1.6.3 (c) OLS and PCR Solutions for the Parameter Matrix B

The estimation errors between the true response variable Y and the estimated response variables $\hat{Y}_{\text{OLS}}$ and $\hat{Y}_{\text{PCR}}$ have been calculated. The Mean Squared Error (MSE) values for these comparisons are as follows:

For OLS Estimation:

$$\text{MSE}_{\text{OLS}} = 0.7103$$

For PCR Estimation:

$$\text{MSE}_{\text{PCR}} = 0.7154$$

The MSE measures the average squared difference between the true values Y and the estimated values $\hat{Y}_{\text{OLS}}$ and $\hat{Y}_{\text{PCR}}$. These values indicate the accuracy of the regression models in predicting Y based on the input matrix X_{noise}.

The slightly higher MSE value for PCR ($\text{MSE}_{\text{PCR}} = 0.7154$) compared to OLS ($\text{MSE}_{\text{OLS}} = 0.7103$) suggests that the OLS method may provide slightly better predictions for the given dataset. However, it's important to note that the difference in MSE values is relatively small, indicating that both methods perform reasonably well in estimating Y.

When estimating data from the test set using regression coefficients computed from the training set, we assess how well the regression models generalize to new, unseen data. In this context, we compare the performance of two regression methods: Ordinary Least Squares (OLS) and Principal Component Regression (PCR).

The Mean Squared Error (MSE) values for these comparisons are as follows:

For OLS Estimation on Test Set:

$$\text{MSE}_{\text{test-OLS}} = 0.4755$$

For PCR Estimation on Test Set:

$$\text{MSE}_{\text{test-PCR}} = 0.4703$$

The MSE measures the average squared difference between the true values Y_{test} and the estimated values $\hat{Y}_{\text{test-OLS}}$ and $\hat{Y}_{\text{test-PCR}}$.
These values reflect the predictive performance of the regression models when applied to the test set data.

Interestingly, the PCR method ($\text{MSE}_{\text{test-PCR}} = 0.4703$) demonstrates slightly better predictive performance on the test set compared to OLS ($\text{MSE}_{\text{test-OLS}} = 0.4755$). The lower MSE for PCR suggests that it provides more accurate predictions for Y_{test} when using the regression coefficients computed from the training set.

1.6.4 (d) Validating OCR and PCR Methods

The Mean Square Error (MSE) estimates for the regression models are as follows:

For OLS Solution:

$$\text{MSE}_{\text{OLS}} = 0.4495$$

For PCR Solution:

$$\text{MSE}_{\text{PCR}} = 0.4697$$

The MSE measures the average squared difference between the true values Y and the estimated values $\hat{Y}$. In this context, a lower MSE indicates better predictive performance. Comparing the MSE values, we observe that the OLS solution ($\text{MSE}_{\text{OLS}} = 0.4495$) exhibits slightly better performance in estimating Y compared to the PCR solution ($\text{MSE}_{\text{PCR}} = 0.4697$) on the test data.

In summary, the OLS method demonstrates slightly superior predictive accuracy, as indicated by the lower MSE value, when applied to the test data.

2 Adaptive Signal Processing

2.1 The Least Mean Square (LMS) Algorithm

2.1.1 (a) Entries of Correlation Matrix

Given a second-order auto-regressive (AR) process as follows:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n), \tag{10}$$

where $\eta(n) \sim \mathcal{N}(0, \sigma_\eta^2)$ represents white Gaussian noise with zero mean and variance σ_η^2. The coefficients of the AR process are specified as $a_1 = 0.1$, $a_2 = 0.8$, and the noise variance as $\sigma_\eta^2 = 0.25$.

To approximate the AR coefficients a_1 and a_2 using the signal $x(n)$, a LMS estimator that functions as a dynamic system is employed. This system treats the signal and its immediate predecessor as inputs, arranged in the vector form $x(n) = [x(n), x(n-1)]^T$. The autocorrelation matrix R_{xx} of the input vector, essential for this estimation, is given as follows:

$$R_{xx} = \begin{bmatrix} E[x(n)x(n)] & E[x(n)x(n-1)] \\ E[x(n-1)x(n)] & E[x(n-1)x(n-1)] \end{bmatrix} = \begin{bmatrix} r_x(0) & r_x(1) \\ r_x(1) & r_x(0) \end{bmatrix}, \tag{11}$$

Considering the system to be WSS (assuming the signal $x(n)$ exhibits wide-sense stationarity (WSS) properties), the autocorrelation function $r_x(\tau)$ depends only on the lag τ and exhibits symmetry.
Thus, the autocorrelation matrix R_{xx} for the input vector can be represented as above, where $r_x(0)$ and $r_x(1)$ are the autocorrelation values at lags 0 and 1, respectively.

To derive these autocorrelation values, we utilize the Yule-Walker equations, which in this context are given by:

$$r_x(0) = a_1 r_x(1) + a_2 r_x(2) + \sigma_\eta^2, \tag{12}$$
$$r_x(1) = a_1 r_x(0) + a_2 r_x(1), \tag{13}$$
$$r_x(2) = a_1 r_x(1) + a_2 r_x(0). \tag{14}$$

Solving the Yule-Walker equations, we find $r_x(0) = \frac{25}{27}$ and $r_x(1) = \frac{25}{54}$. Therefore, the autocorrelation matrix R_{xx} is approximated as:

$$R_{xx} \approx \begin{bmatrix} 0.926 & 0.463 \\ 0.463 & 0.926 \end{bmatrix}. \tag{15}$$

The LMS algorithm converges in the mean if the step size μ is chosen within a specific range determined by the eigenvalues of the autocorrelation matrix R_{xx}. The convergence criterion is:

$$0 < \mu < \frac{2}{\lambda_{\max}}, \tag{16}$$

where $\lambda_{\max}$ is the largest eigenvalue of R_{xx}. For the given matrix, $\lambda_{\max} = 1.389$. Therefore, the step size μ must satisfy:

$$0 < \mu < 1.44. \tag{17}$$

This range ensures that the LMS algorithm converges to the optimal solution in the mean.

2.1.2 (b) Implementing the LMS Adaptive Predictor

In the context of predicting a signal modeled by an autoregressive (AR) process as given in Eq. 10, the Least Mean Squares (LMS) algorithm seeks to find the filter coefficients that best predict the current sample based on previous samples.

Here, the LMS algorithm on MATLAB to predict samples from an AR process defined by specific coefficients and driven by white Gaussian noise is applied.
The algorithm's performance using two different step sizes ($\mu = 0.05$ and $\mu = 0.01$) was investigated to understand how the rate of adaptation affects prediction accuracy and the results are shown in Fig. 21 and Fig. 22 below.

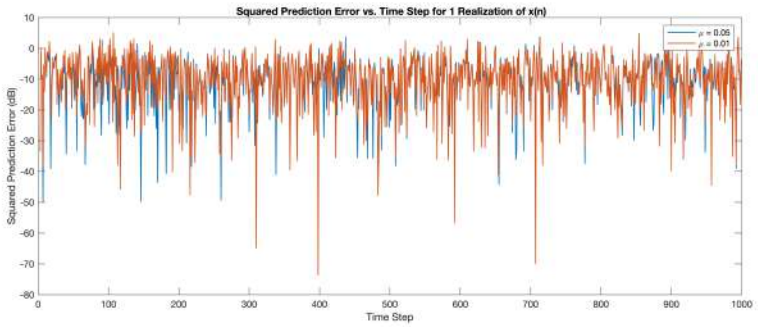

Figure 21: Plot of Squared Prediction Error vs. Time Step for 1 realization of AR process given in Section 2.1.1

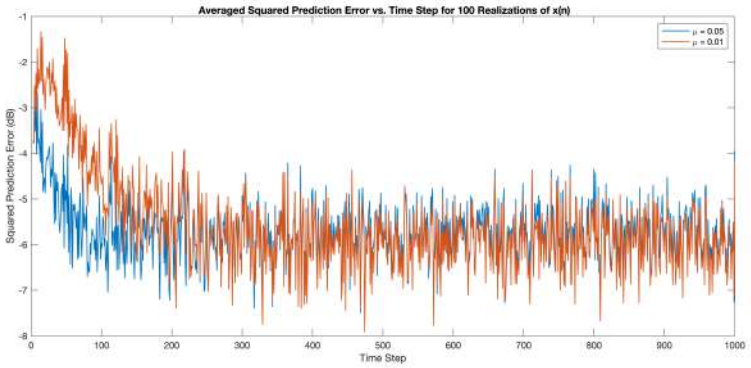

Figure 22: Plot of Averaged Squared Prediction Error vs. Time Step for 100 realizations of AR process given in Section 2.1.1

Observing the squared prediction error from a singular realization as in Fig 21, revealed substantial variability. This makes it difficult to clearly determine if the algorithm is consistently getting better or how much the step sizes affect the outcome. This variability is due to the random nature of the white Gaussian noise driving the AR process.

When looking at the plot for 100 realizations in Fig. 22, a clearer picture of the algorithm's overall behavior was obtained, smoothing out the anomalies presented by individual noise instances.
This result highlighted a notable distinction in convergence speeds between the two step sizes. Specifically, a step size of $\mu = 0.05$ demonstrated a quicker convergence, reaching a stable error level after approximately 100 samples. In contrast, the smaller step size of $\mu = 0.01$ exhibited a more gradual convergence, stabilizing after roughly 200 samples.

Interestingly, while the larger step size facilitated faster convergence, it also settled at a higher steady-state error compared to the smaller step size. This outcome illustrates a fundamental trade-off inherent in the selection of the step size for adaptive LMS algorithms.

2.1.3 (c) Calculation of Theoretical and Estimated LMS Misadjustment

The misadjustment metric, M, is crucial for assessing the performance of the LMS algorithm in adaptive filtering. It quantifies the excess error introduced due to the adaptive filter's weight fluctuations around the optimal solution.
The theoretical and experimental LMS misadjustment values for $\mu = 0.05$ and $\mu = 0.01$ are given as:

Step Size (μ)	Theoretical Misadjustment ($M_{Theoretical}$)	Experimental Misadjustment ($M_{Estimated}$)
0.05	0.0463	0.0446
0.01	0.0093	0.0043

Table 2: Comparison of Theoretical and Experimental Misadjustment Values.

Calculating Theoretical Misadjustment:

The theoretical misadjustment for the LMS algorithm, given a small step size (μ), is approximated by the formula:

$$M_{Theoretical} \approx \mu \times \frac{Tr\{R\}}{2}, \tag{18}$$

where $Tr\{R\}$ denotes the trace of the input signal's autocorrelation matrix R given in Eq.15, and μ is the learning rate or step size. The trace of matrix R, $Tr\{R\}$, is the sum of its diagonal elements, providing insight into the signal's power.

Calculating Estimated Misadjustment:

The experimental misadjustment is derived from empirical observations, using the formula:

$$M_{Estimated} = \frac{EMSE}{\sigma_\eta^2}, \tag{19}$$

where EMSE (Excess Mean Square Error) measures the additional error power introduced by the adaptive filter over the minimum achievable error (σ_η^2). This metric is obtained by analyzing the steady-state portion (The last 20% of time step) of the ensemble-averaged squared prediction error over 100 realizations.

The close alignment between theoretical and experimental misadjustment values underscores the LMS algorithm's robustness.
Notably, the experimental misadjustment for $\mu = 0.05$ only slightly deviates from the theoretical prediction, suggesting efficient adaptation with minimal excess error. Conversely, the significantly lower experimental misadjustment for $\mu = 0.01$ than its theoretical counterpart highlights enhanced precision and stability at smaller step sizes.
These findings validate the theoretical models' accuracy in predicting LMS performance and emphasize the impact of step size selection on adaptation quality and error convergence.

2.1.4 (d) Estimating Steady State Values of the Adaptive Filter Coefficients

The estimated steady-state coefficients were computed by averaging their values over the identified steady-state region across all trials. This approach minimized the influence of outlier trials and provided a robust estimate of the LMS algorithm's performance in adapting to the AR(2) process.

The time evolution of these averaged coefficients was plotted against the true values, for both step sizes, to visually assess the convergence behavior and accuracy of the LMS algorithm. The results are shown in Fig.23.

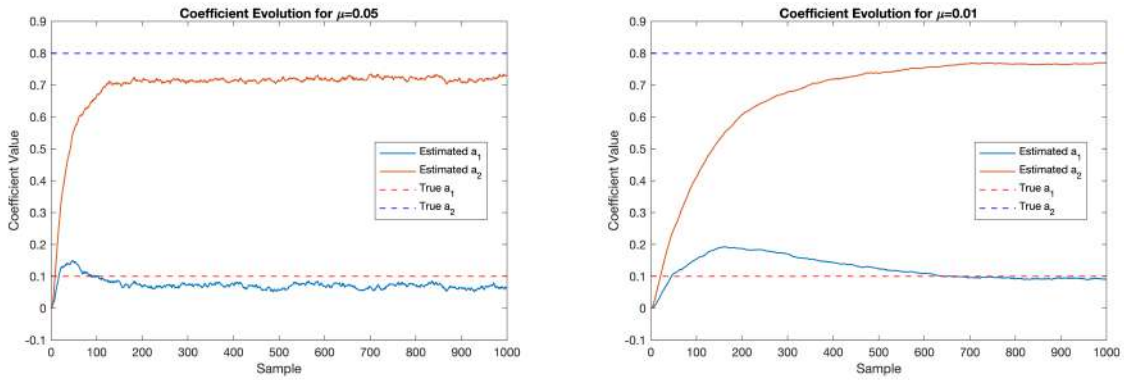

Figure 23: Time evolution of adaptive filter coefficients for $\mu = 0.05$ and $\mu = 0.01$.

From Fig.23 we observe that for $\mu = 0.05$, the estimated a_1 is 0.065 and the estimated a_2 is 0.7184, while for $\mu = 0.01$, the estimated a_1 is 0.096 and the estimated a_2 is 0.766.

The larger step size ($\mu = 0.05$) facilitated faster convergence of the filter coefficients towards their steady-state values as seen by the steep gradient of the curves on the left plot of Fig.23. This behavior aligns with theoretical

expectations, as a larger step size allows for more significant adjustments to the coefficients with each iteration.

While both step sizes enabled the algorithm to approximate the true coefficients, the smaller step size ($\mu = 0.01$) resulted in closer estimates to the true values. This improved accuracy can be attributed to the finer adjustments made to the coefficients, allowing for a more precise convergence.
As a result, the choice of step size presents a trade-off between convergence speed and estimation accuracy.

2.1.5 (e) The Leaky LMS Algorithm - The Leaky LMS Equation Proof

The Leaky LMS algorithm introduces a regularization term or leakage coefficient (γ) to stabilize the weight adaptation process, particularly when the input autocorrelation matrix exhibits eigenvalues that are zero. The algorithm achieves stability by employing a cost function defined as:

$$J_2(n) = \frac{1}{2}e^2(n) + \frac{\gamma}{2}\|w(n)\|_2^2, \tag{20}$$

where $e(n) = d(n) - y(n)$ denotes the prediction error with $y(n)$ as the predicted output and $d(n)$ as the desired output. The term $\|w(n)\|_2^2$ corresponds to the squared Euclidean norm of the weight vector $w(n)$, and γ is a small positive constant known as the leakage coefficient.

The minimization of $J_2(n)$ leads to the formulation of a weight update rule that integrates the leakage factor. The update rule can be derived by computing the gradient of $J_2(n)$ with respect to the weight vector and applying gradient descent:

$$\nabla_{w(n)} J_2(n) = \nabla_{w(n)} \left(\frac{1}{2}e^2(n) + \frac{\gamma}{2} w^T(n)w(n) \right) \tag{21}$$

$$= -e(n)x(n) + \gamma w(n). \tag{22}$$

Incorporating the gradient into the update rule, with the learning rate μ, results in:

$$w(n+1) = w(n) - \mu \nabla_{w(n)} J_2(n) \tag{23}$$

$$= w(n) + \mu e(n)x(n) - \mu\gamma w(n) \tag{24}$$

$$= (1 - \mu\gamma)w(n) + \mu e(n)x(n). \tag{25}$$

Thus, we obtain the leaky LMS update equation as:

$$w(n+1) = (1 - \mu\gamma)w(n) + \mu e(n)x(n), \tag{26}$$

which illustrates that the adaptation of weights in the leaky LMS algorithm is indeed directed by the steepest descent of the modified cost function J_2.

2.1.6 (f) The Leaky LMS Algorithm - Implementing the Leaky LMS Algorithm

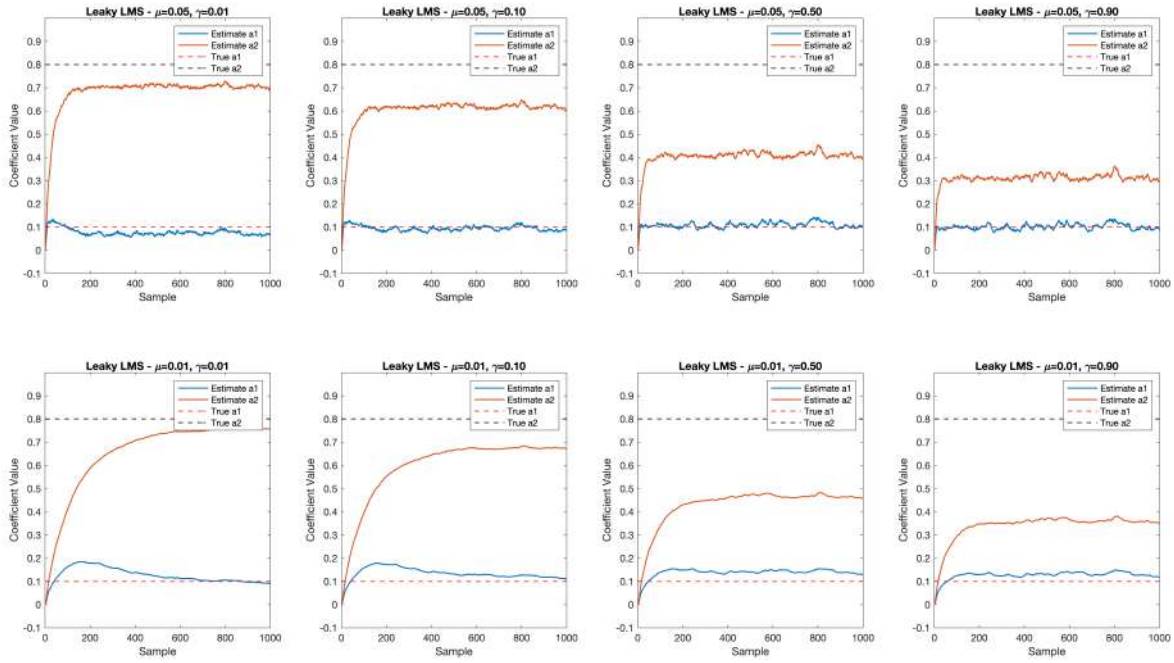

Figure 24: Estimated AR(2) coefficients using Leaky LMS algorithm for different values of γ and μ.

Leaky LMS algorithm was utilised to find the AR coefficients for process defined in part 2.1.1 for different values of γ and μ. The results are shown in Fig.24.

The Leaky-LMS algorithm introduces a leakage coefficient γ to stabilize the LMS algorithm when the autocorrelation matrix R_{xx} of the input signal possesses zero eigenvalues, which may cause instability due to a positive semi-definite matrix. This matrix modification forces the filter weights to converge by shifting R_{xx} from positive semi-definite to positive definite, thereby ensuring invertibility. The updated weight vector in the Leaky-LMS algorithm is given by the expression:

$$w_{optLeaky} = (R_{xx} + \gamma I)^{-1} p, \tag{27}$$

compared to the standard Wiener-Hopf equation, which is:

$$w_{opt} = R_{xx}^{-1} p, \tag{28}$$

where p is the cross-correlation vector between the input vector $x(n)$ and the desired output $d(n)$. As the leakage coefficient γ tends toward zero, the Leaky-LMS weights mean-converge to the Wiener-Hopf solution, indicating that the regularization effect induced by the leakage term becomes negligible.

The weights obtained from the Leaky-LMS algorithm might not converge to the true AR process coefficients due to several reasons:

Regularization Effect: The leakage term adds a regularization effect to the LMS adaptation process. While this regularization can be advantageous in scenarios where the input exhibits near-zero eigenvalues—preventing overfitting—it may introduce bias into the weight estimates when not necessary, moving them away from the true system parameters as seen for greater values of γ when μ remains constant in Fig.24.

Inappropriate γ Value: Selecting a leakage coefficient γ that is too large can result in excessive penalization of the weight magnitudes. This excessive penalty causes the weights to converge to values that minimize the regularized cost function, thereby deviating from the true parameter values.

Step Size and Leakage Coefficient Interaction: The relationship between the step size μ and the leakage coefficient γ is critical. An unsuitable combination of these parameters can cause sub-optimal convergence, characterized by either slow adaptation or an overshoot of the true coefficients.

To achieve precise weight estimation, the leakage coefficient γ should be chosen carefully. For input signals without near-zero eigenvalues in the autocorrelation matrix, a smaller γ, close to zero, is generally recommended to enhance weight estimation accuracy, balancing between convergence speed, algorithm stability, and precision.

2.2 Adaptive Step Sizes

2.2.1 (a) The GASS Algorithms

The performance of the GASS (Gradient Adaptive Step Size) algorithms on identifying a real valued MA(1) process as compared to standard fixed LMS is shown in Fig. 25.

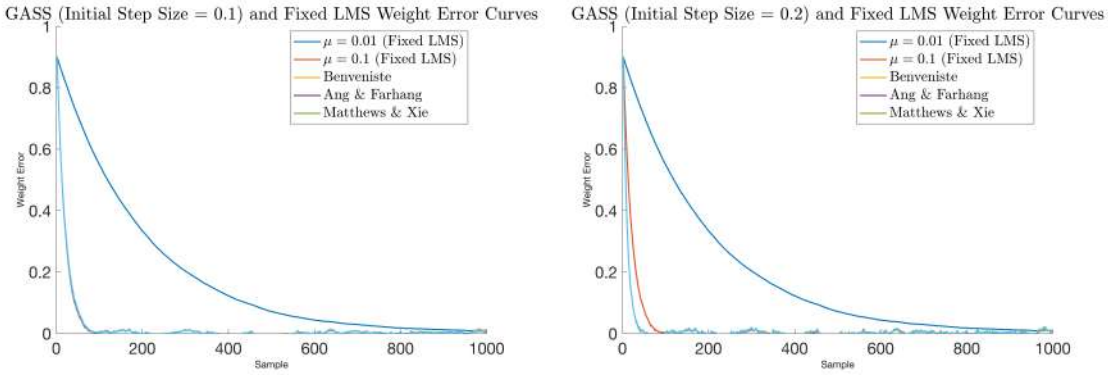

Figure 25: Weight error curves for GASS algorithm with 2 initial step sizes compared with standard LMS.

The Fixed LMS algorithm, with its ease of implementation, is widely used for adaptive filtering and system identification. The fixed LMS algorithm updates the weight vector using the equation:

$$w(n+1) = w(n) + \mu(n) e(n) x(n) \tag{29}$$

where $w(n)$ is the weight vector at time n, μ(n) is the step size, $e(n)$ is the error between the desired and the actual output, and $x(n)$ is the input vector. The choice of μ presents a trade-off: a larger μ leads to faster convergence but can result in higher steady-state error, while a smaller μ reduces steady-state error but slows down convergence.

GASS algorithms, including Benveniste, Ang & Farhang, and Matthews & Xie as shown in both plots of Fig.25, adapt the step size $\mu(n)$ dynamically to improve the algorithm's performance over the standard LMS.
These adaptive methods aim to balance the trade-off between convergence speed and steady-state error by adjusting $\mu(n)$ based on the algorithm's current state. The update equations for $\mu(n)$ in these algorithms are designed to minimize both the cost function with respect to the weight vector and the cost function with respect to the step size simultaneously.

Advantages:

- **Faster Convergence**: By increasing the step size during the initial phase (0.1 in left plot anf 0.2 in right plot of Fig.25), all 3 GASS algorithms converge faster to the optimal solution compared to standard LMS with a fixed small step size. Benveniste algorithm is seen to converge the fastest in both plots followed by Ang & Farhang, and then by Matthews & Xie.
- **Reduced Steady-State Error**: As the algorithms approach the optimal solution, they decrease the step size, reducing the steady-state error.

Disadvantages:

- **Complexity**: GASS algorithms are more complex to implement and require more computational resources than the standard LMS.
- **Parameter Tuning**: Additional parameters (e.g., ρ and α) need to be tuned, which can be non-trivial. The GASS algorithms as shown in Fig. 25 involved hyperparamters of $\rho = 0.01$ and $\alpha = 0.8$.

The Normalized LMS (NLMS) algorithm introduces a normalization factor based on the power of the input signal, addressing the sensitivity of the standard LMS to the input signal's scale. The NLMS update equation is:

$$w(n+1) = w(n) + \frac{\beta e(n)x(n)}{\epsilon + \|x(n)\|^2} \tag{30}$$

where β is the step size, ϵ is a small constant to avoid division by zero, and $\|x(n)\|^2$ is the squared norm of the input vector, providing the normalization factor.

Advantages:

- **Robustness to Input Scale**: NLMS adapts to the variance of the input, making it more robust to changes in input signal scale.
- **Improved Convergence**: The normalization can lead to faster convergence in varied signal conditions without significantly increasing the steady-state error.

Disadvantages:

- **Parameter Choice**: The choice of ϵ and β can affect performance, requiring careful tuning.
- **Slightly Higher Complexity**: The need to calculate the squared norm of the input vector at each iteration slightly increases computational complexity compared to standard LMS.

2.2.2 (b) Equivalence of LMS Update Equation to NLMS

To demonstrate the equivalence of the LMS update equation, based on a posteriori error, to the Normalized LMS (NLMS) algorithm, we initiate with the update formula:

$$w(n+1) = w(n) + \mu e_p(n)x(n) \tag{31}$$

where the a posteriori error is $e_p(n) = d(n) - x^T(n)w(n+1)$. The priori error, defined as $e(n) = d(n) - x^T(n)w(n)$, is the starting point for our derivation.

Multiplying both sides of the equation by $x^T(n)$ and rearranging, we obtain:

$$x^T(n)w(n+1) = x^T(n)w(n) + \mu e_p(n)\|x(n)\|^2 \tag{32}$$

By incorporating $d(n)$ into both sides and expressing in terms of $e_p(n)$, it simplifies to:

$$e_p(n) = e(n) - \mu e_p(n)\|x(n)\|^2 \tag{33}$$

Solving for $e_p(n)$ in terms of $e(n)$ yields:

$$e_p(n) = \frac{e(n)}{1 + \mu\|x(n)\|^2} \tag{34}$$

This formulation allows us to express the weight update difference $\Delta w(n) = w(n+1) - w(n)$ as:

$$\Delta w(n) = \mu e(n) \frac{x(n)}{1 + \mu\|x(n)\|^2} \tag{35}$$

Given the NLMS update rule:

$$\Delta w_{NLMS}(n) = \frac{\beta e(n) x(n)}{\epsilon + \|x(n)\|^2} \tag{36}$$

Comparing this derived expression with the NLMS update rule in Eq.35, we observe that the formulations align when setting $\beta = \mu$ and $\epsilon = \frac{1}{\mu} - 1$. This demonstrates the equivalence of the LMS algorithm based on a posteriori error to the NLMS algorithm under specific parameter conditions.

2.2.3 GNGD Algorithm

Fig.26 below illustrates the time evolution of weight estimates obtained from GNGD and Benveniste algorithms. The theoretical weight for this system was 0.9 as given in section 2.2.1. The weight estimates represent the adaptation of the algorithms to the system's dynamics.
The optimal hyperparameters used for GNGD are $\rho = 0.05$ and $\mu = 1$, while for Benveniste algorithm the values were $\rho = 0.02$ and $\mu = 0.1$. Grid search was used a hyperparamter tuning method, where each combination of hyperparameters is evaluated. The combination of ρ and μ that gave the least square prediction error was used to plot the weight estimates.
From Fig.26 we observe that GNGD algorithm has an overall faster convergence than Benveniste algorithm.

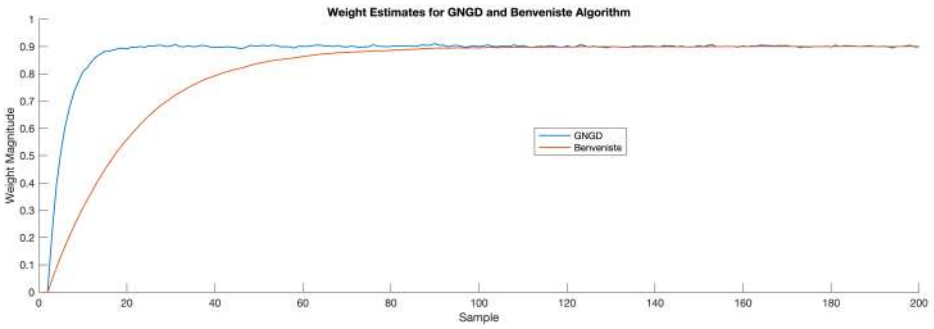

Figure 26: Weight estimates curve for GNGD and Benveniste's algorithms for their respective optimal hyperparamter values.

For Benveniste's algorithm, the complexity mainly arises from the update of the sensitivity vector $\psi(n)$ and the weight vector $w(n+1)$. The complexity grows quadratically with the model order N_w due to the outer product calculation in the update of $\psi(n)$.

Operation	**Multiplications/Divisions**	**Additions/Subtractions**
A: $e(n-1)x(n-1)$	N_w	0
B: $\mu(n-1)x(n-1)x^T(n-1)$	$N_w^2 + N_w$	$N_w(N_w - 1)$
C: $(I - B)\psi(n-1) + A$	N_w^2	$N_w^2 + N_w$
D: $w^T(n)x(n)$ and $e(n) = d(n) - D$	N_w	$N_w - 1$
E: $x^T(n)\psi(n)$ and update $\mu(n) + \rho e(n)E$	$N_w + 1$	N_w
F: $w(n) + \mu(n)e(n)x(n)$	N_w	N_w
Total	$3N_w^2 + 4N_w + 1$	$2N_w^2 + 3N_w - 1$
Overall Complexity	$O(N_w^2)$	

Table 3: Comprehensive computational complexity for updating $\psi(n)$ (A,B,C) and $w(n+1)$ (F) in Benveniste's algorithm, illustrating the quadratic growth in complexity with respect to the model order N_w.

For the GNGD algorithm, the computational complexity is primarily due to the updates of $w(n+1)$ and $\epsilon(n+1)$, with a complexity that grows linearly with the model order N_w.

Operation	Multiplications/Divisions	Additions/Subtractions
A: $\|x(n)\|^2$	N_w	$N_w - 1$
B: $\beta e(n)$ over $\epsilon(n) + A$	2 (for division and multiplication)	1 (for addition in denominator)
C: $w(n) + Bx(n)$	N_w	N_w
D: $\|x(n-1)\|^2$	N_w	$N_w - 1$
Total for $w(n+1)$	$4N_w + 2$	$4N_w + 1$
Operation for $\epsilon(n+1)$	9	3
Total	$4N_w + 11$	$4N_w + 3$
Overall Complexity	$O(N_w)$	

Table 4: Comprehensive computational complexity for the GNGD algorithm. This table illustrates the overall growth in complexity with respect to the model order N_w, indicating a linear relationship.

This comparison indicates that for large model orders, Benveniste's algorithm incurs significantly higher computational costs compared to the GNGD algorithm. This discrepancy in computational demand is primarily attributed to the operation $x(n-1)x^T(n-1)$ involved in updating the sensitivity vector $\psi(n)$ in Benveniste's approach.
The outer product calculation $x(n-1)x^T(n-1)$ leads to a quadratic increase in computational complexity as the model order increases, whereas the GNGD algorithm maintains a linear growth in complexity with respect to the model order N_w.

2.3 Adaptive Noise Cancellation

2.3.1 Delay for Adaptive Line Enhancer (ALE)

Given a signal $s(n) = x(n) + \eta(n)$, where $x(n) = \sin(0.01\pi n)$ represents a deterministic clean signal and $\eta(n)$ denotes colored noise modeled by $\eta(n) = v(n) + 0.5v(n-2)$ with $v(n) \sim \mathcal{N}(0,1)$, the goal is to identify the minimum delay Δ that could be employed in an Adaptive Line Enhancer (ALE) with a filter length $M > 1$.

The mean squared error (MSE) can be expanded as follows:

$$\mathbb{E}\left[(s(n) - \hat{x}(n))^2\right] = \mathbb{E}\left[((x(n) - \hat{x}(n)) + \eta(n))^2\right] \tag{37}$$

Decomposing the MSE yields:

$$\mathbb{E}\left[(x(n) - \hat{x}(n))^2\right] + 2\mathbb{E}\left[\eta(n)x(n)\right] - 2\mathbb{E}\left[\eta(n)\hat{x}(n)\right] + \mathbb{E}\left[\eta^2(n)\right] \tag{38}$$

With $x(n)$ being deterministic and $\eta(n)$ having zero mean, the cross-term involving $x(n)$ vanishes. The variance of $\eta(n)$, $\mathbb{E}\left[\eta^2(n)\right]$, equates to 1.25. To minimize the MSE in relation to the noise's correlation properties, attention is directed towards the term $-2\mathbb{E}\left[\eta(n)\hat{x}(n)\right]$.
Given for ALE that:

$$\hat{x}(n) = \mathbf{w}^T(n)\mathbf{u}(n) \tag{39}$$

$$\mathbf{u}(n) = \left[s(n-\Delta), s(n-\Delta-1), \ldots, s(n-\Delta-M+1)\right]^T \tag{40}$$

Investigating the correlation term between $\eta(n)$ and $\hat{x}(n)$, and considering the autocorrelation function (ACF) of $\eta(n)$, $E\left[\eta(n)\eta(n-k)\right] = 2\delta(k) + 0.5\delta(k-2)$, it is discerned that the minimum delay $\Delta_{\min} = 3$. This delay ensures the term $-2\mathbb{E}\left[\eta(n)\hat{x}(n)\right]$ is nullified, minimizing the MSE by eliminating the influence of noise correlations.

To justify the above calculation an ALE LMS algorithm of step size $\mu = 0.01$ (to allow for convergence) was implemented on MATLAB, and MSPE values for different Delays Δ for Filter order of 5 were plotted. The results align with the theoretical calculations above and are shown in Fig.27.

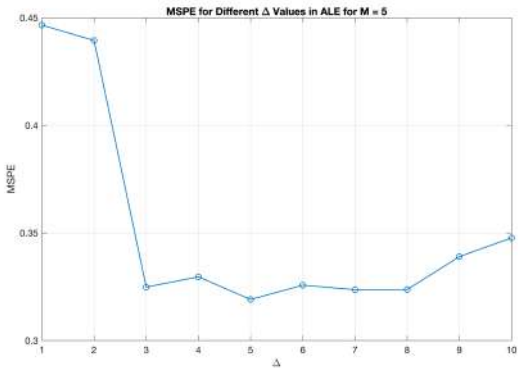

Figure 27: MSPE values against different Delay for Filter order of 5 using the ALE LMS algorithm

2.3.2 Dependance of MSPE on Δ and Filter Order M

The ALE-LMS algorithm implemented in part 2.3.1 was used to plot a graph of MSPE against different Δ for different values of filter order M. The results are shown in Fig.28 (Left).

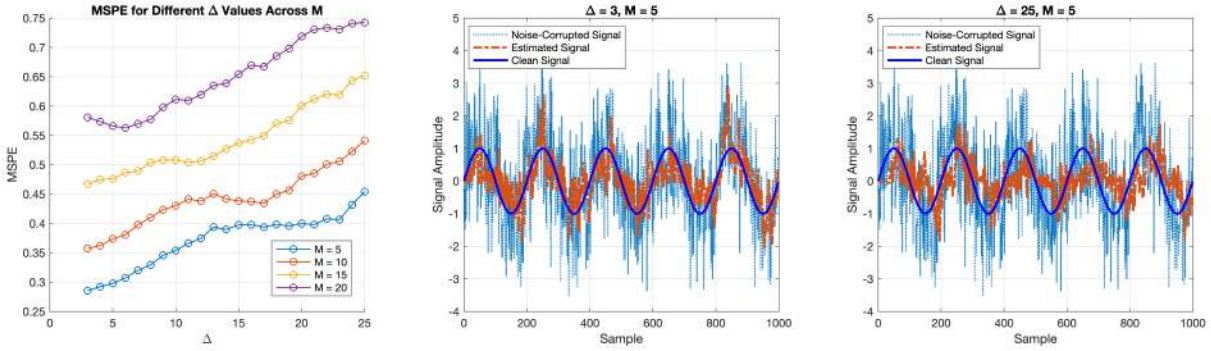

Figure 28: MSPE values against different Delay for different Filter order of 5 using the ALE LMS algorithm (Left). Estimate of x(n) for $\Delta = 3$ and $M = 5$ (Middle).Estimate of x(n) for $\Delta = 25$ and $M = 5$ (Right).

We observe that for smaller values of Δ, especially those close to the minimum Δ determined theoretically, the MSPE tends to be higher. As Δ increases from this minimum value, there is generally an initial decrease in MSPE. This improvement occurs because increasing Δ helps in decorrelating the noise component in the predictor inputs from the current noise, thereby enhancing the ALE's ability to suppress the noise and better estimate the clean signal.

From Fig.28 (Left) we observe this for $\Delta \leq 2$ the MSPE for all model orders has a high value, which decreases as Δ reaches a value of 3. From the figure we also observe the optimal hyperparameters are $M = 5$ and $\Delta = 3$ which give us the lowest MSPE. This is also shown in Fig.28 (Middle) where the estimated signal is plotted and closely matches the clean signal.

Beyond the optimal Δ range, the MSPE might start to increase again as seen in Fig.28(Left) after $\Delta = 6$. This increase can occur because a too-large delay could lead the ALE to neglect relevant parts of the signal structure, potentially treating useful signal components as noise due to the excessive delay. From Fig.28 (Left) we also observe the MSPE depends on the filter order M. As the filter order increases beyond the optimal filter older the MSPE begins to increase. A pragmatic choice for M might be the lowest among the values that achieve near-minimal MSPE which in this case is $M = 3$, considering the increasing computational cost with higher M values.

2.3.3 Comparing ALE with ANC (Adaptive Noise Cancellation)

The ANC-LMS algorithm is fundamentally designed to reduce the noise component from a primary signal by using a reference noise signal that is ideally correlated, but not identical, to the actual noise corrupting the primary signal. The reference noise used in this implementation was

$$\epsilon(n) = 2\eta - 0.3 \tag{41}$$

A reference noise signal that maintains a linear correlation with the primary noise enables the ANC to effectively model the noise characteristics and dynamics, facilitating its removal while de-noising the signal.

The ANC-LMS algorithm was implemented on MATLAB and it's results were compared to the ALE-LMS algorithm implemented in part 2.3.1. The results of the de-noised estimated signals are shown in Fig.29.

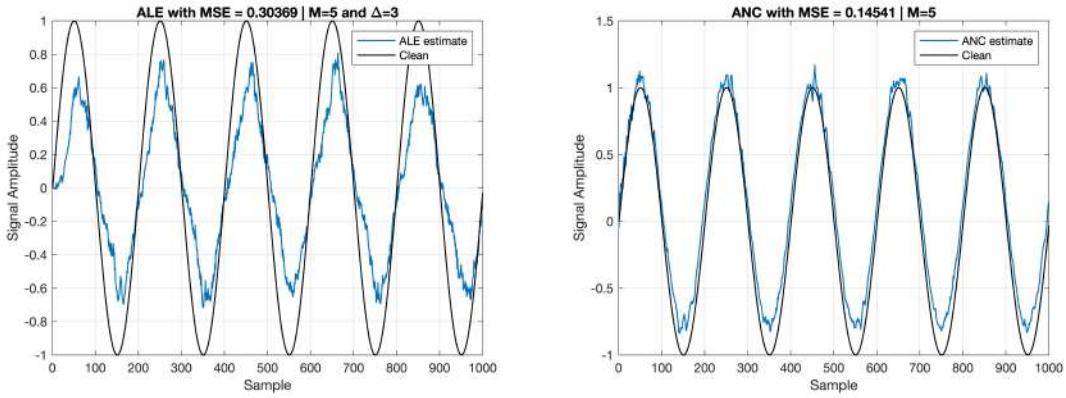

Figure 29: ALE de-noised estimated signal (Left) along with its MSPE for optimal hyperparameters of M = 5 and Δ = 3 compared against ANC de-noised estimated signal (Right) for M = 5.

The ALE graph shows a higher degree of variance between the estimated signal and the clean signal, particularly notable in the amplitude of the oscillations as seen in Fig.29. This indicates that while ALE has some capability to track the clean signal, it does not filter out the noise as effectively, which is consistent with the higher MSPE value of 0.30516.
The ALE algorithm relies on the predictability of the signal itself and is less effective when the signal and noise share similar frequencies. Hence, the ALE performance is somewhat consistent throughout the samples but doesn't achieve a high degree of noise reduction.
The ANC graph displays a closer match between the estimated signal and the clean signal, demonstrating that the ANC algorithm is more successful in suppressing the noise. This is corroborated by the lower MSPE value of 0.14725. The ANC's performance can be attributed to the presence of a reference noise signal that is correlated with the noise in the primary input.

Moreover, when dealing with smaller datasets, ALE outperforms ANC because ALE doesn't require an initial learning phase to converge. ALE can start filtering noise based on the autocorrelation of the signal from the very beginning. In contrast, ANC requires a certain amount of data to learn and adapt to the noise, which may cause suboptimal performance in the initial phase when data size is limited. This is seen as noise (spiking) in the signal amplitude at lower samples in Fig.29.

2.3.4 De-noising EEG Data using ANC

Spectrogram of the noise corrupted EEG data from Cz electrode with Hamming window length of 4800 and overlap of 50% is plotted in Fig.30 below. The strong 50 Hz component is visible clearly in this plot.

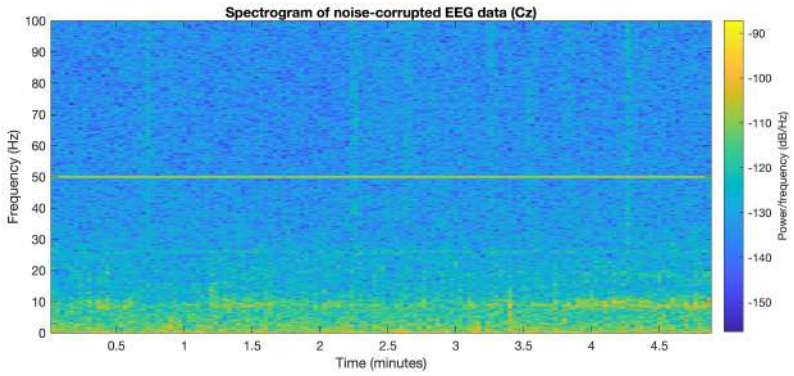

Figure 30: Spectrogram of noise corrupted EEG data (Cz).

To remove the 50 Hz component the ANC algorithm from part 2.3.1 is implemented with a synthetic reference signal comprising of the sinusoidal signal representing the 50 Hz and a zero mean gaussian white noise with standard deviation of 0.001.

The values of filter order M and step size μ are varied to find the optimal hyperparameters that suppress the 50 Hz component without affecting the other frequency components. The results are plotted below in Fig.31.

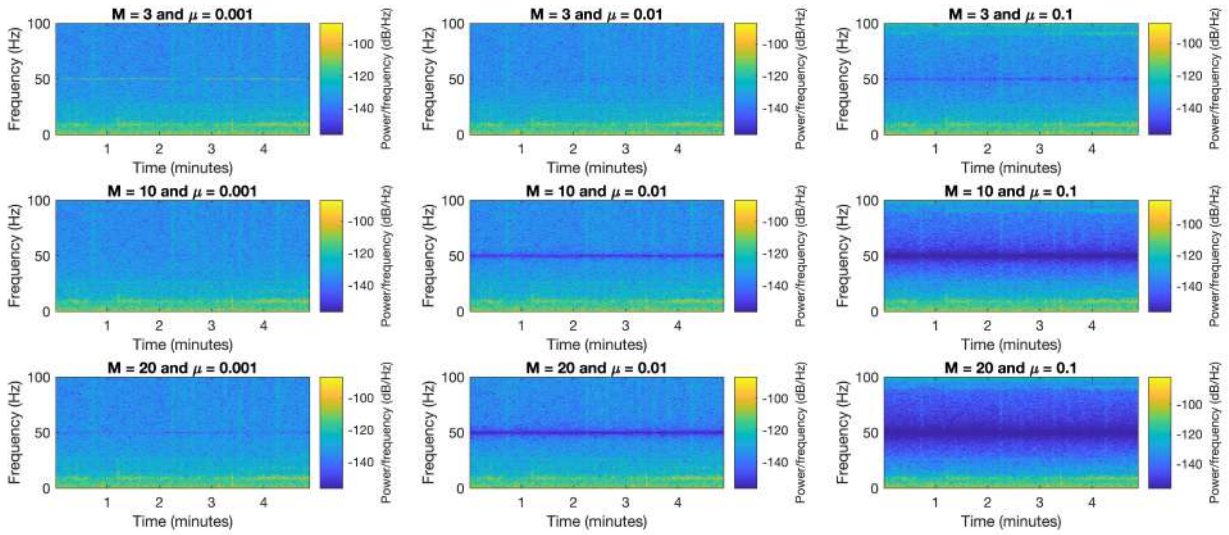

Figure 31: Spectrogram of de-noised EEG Data by implementing ANC algorithm for different values of M and μ.

There is a clear dependency of the ANC effectiveness on the step size. For a small step size ($\mu < 0.001$), the 50 Hz noise is not completely removed, suggesting insufficient adaptation of the filter weights.
As the step size increases, the ability to cancel out the 50 Hz component improves, which can be seen in the reduction of the yellow band at 50 Hz from Fig.31.
However, with a large step size ($\mu = 0.1$), there is a risk of over-adjusting the filter weights, potentially distorting the desired signal components and affecting the other frequency components.

Filter length affects the ANC's capability to model and remove noise. Short filters (M = 3) have less precision in noise estimation and cancellation, while longer filters (M = 10, 20) demonstrate more effective noise removal.
However, excessively long filters may introduce unnecessary complexity and computational load, and may also affect the signal components outside of the targeted noise frequency.

The intermediate values of M = 10 and $\mu = 0.001$) appear to offer a balance between effective noise cancellation at 50 Hz and preserving the integrity of the rest of the EEG spectrum.
Using these hyperparameters the de-noised spectrogram of EEG data is plotted in Fig.32.

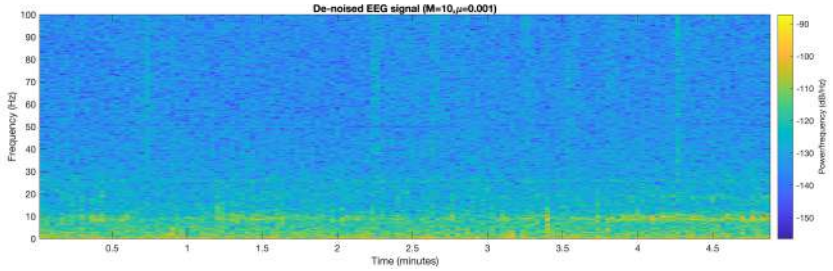

Figure 32: Spectrogram of de-noised EEG Data by implementing ANC algorithm for M =10 and $\mu = 0.001$.

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

A first-order widely-linear-moving-average (WLMA(1)) process defined as:

$$y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-1) \tag{42}$$

where $b_1 = 1.5 + j$ and $b_2 = 2.5 - 0.5j$ and circular white Gaussian noise $x(n)$ was generated CLMS and ACLMS algorithms were implemented on MATLAB to identify the WLMA model.

The circularity coefficient and the complex samples of the WLMA(1) process are plotted. The results are ploted in Fig.33.

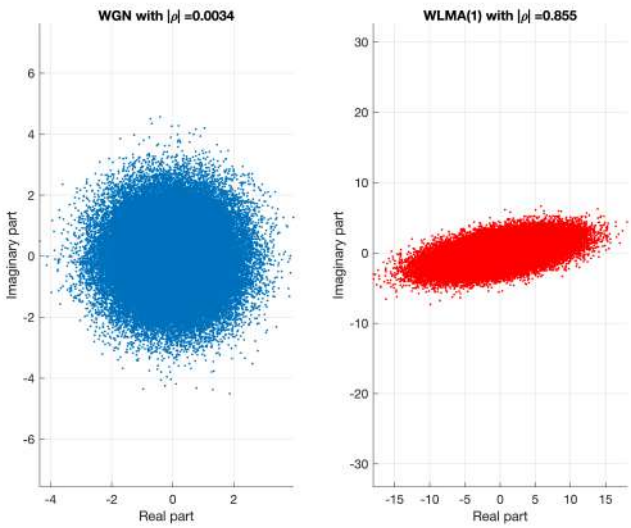

Figure 33: Circularity plot of WGN (Left) and WLMA model (Right) showing the imaginary and real values of the data.

The circularity coefficient is a vital parameter in signal processing, quantifying the degree of noncircularity of a complex random variable. It is defined by the formula:

$$|\rho| = \frac{|E\{zz^*\}|}{E\{|z|^2\}} \tag{43}$$

where z is a complex random variable, $E\{\cdot\}$ denotes the expected value, zz^* is the pseudocovariance, and $|z|^2$ is the actual covariance of z.
Noncircular signals like the WLMA(1) process cannot be fully characterized by their first-order moments (mean and variance) alone. Instead, their second-order structure also includes pseudocovariance, which captures the correlation between a variable and its complex conjugate.

The plot on the left in Fig.33 illustrates the complex samples of WGN.This uniform scattering is characteristic of circular noise, where the probability distribution of the noise is rotationally invariant. The circularity coefficient $\rho \sim 0$ indicates that the noise is indeed nearly perfectly circular, confirming that the real and imaginary parts of the noise are uncorrelated and have equal variance.
The plot on the right of Fig.33 shows the complex samples of the WLMA(1) process. This elliptical distribution is due to the WLMA process being influenced by its past values, both direct and conjugated. The non-zero circularity coefficient $\rho \sim 0.855$ suggests a high degree of noncircularity, meaning that there is a significant correlation between the real and imaginary parts of the process or an unequal variance between them.

The learning curve for CLMS and ACLMS algorithm was then plotted and the results are shown in Fig.34.

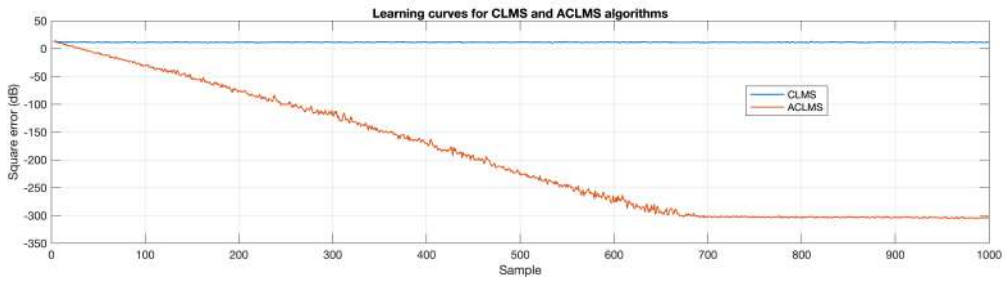

Figure 34: Learning curve for CLMS and ACLMS algorithm.

This plot demonstrates the superior performance of the ACLMS algorithm over the CLMS algorithm for the system identification task with noncircular complex data. The results provide empirical evidence supporting the theoretical expectation that ACLMS should perform better than CLMS when dealing with signals that have noncircular properties, as it can exploit the full second-order statistics of the signal. This is evidenced by the lower steady-state error of the ACLMS, indicating a closer match to the system being identified.

From the plot we observe that both algorithms start with a high level of error. This is normal as the algorithms begin with initial weights that are likely far from the optimal values needed to minimize error.
As the number of samples increases, both algorithms converge towards a steady state. The CLMS algorithm appears to reach a plateau quickly, maintaining a consistent error level of around 11.6 dB.
The ACLMS, on the other hand, continues to decrease in error over a more extended period before reaching its steady state. This suggests that the ACLMS is capable of extracting additional information from the data, which allows it to achieve a lower error compared to the CLMS algorithm of around -306 dB.

3.1.1 (b) Assessing Bivariate Wind Data

The three wind regimes (low, medium, high) were formed into a complex valued wind signal given by

$$v[n] = v_{\text{east}}[n] + jv_{\text{north}}[n] \tag{44}$$

The circularity plot for the 3 wind regimes are shown in Fig.35 below.

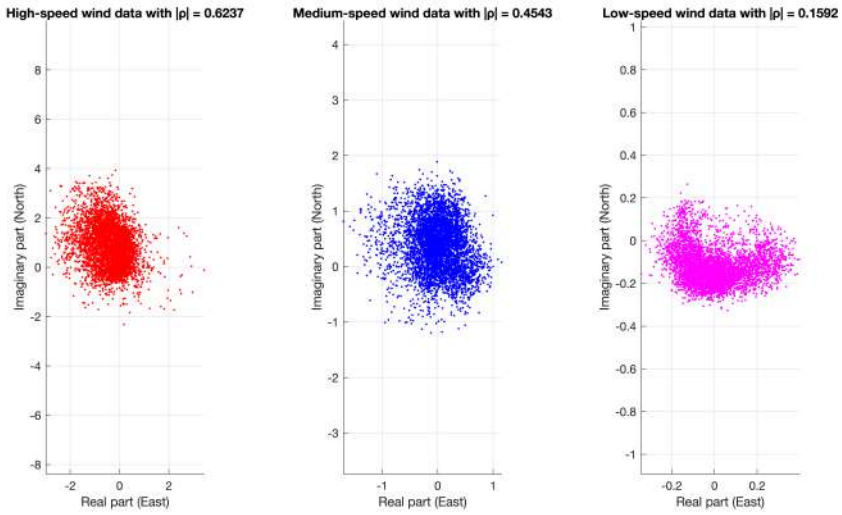

Figure 35: Circulatory plot for High (Left), Meidum (Middle) and Low (Right) wind speed data.

The scatter plots depict the distribution of wind data in the complex plane, with the real part corresponding to the east-west (E-W) wind component and the imaginary part to the north-south (N-S) component. The circularity coefficient ρ quantifies the degree of non-circularity in the data.
For circular or proper complex random variables, this coefficient is zero because the distribution is rotationally invariant, meaning that the statistical properties of the data do not change with rotation in the complex plane.

In the high-speed wind data plot, the spread indicates variability in wind speed, and the elongation along the imaginary axis suggests a stronger variance in the N-S component compared to the E-W component. The moderate ρ value reflects a significant degree of non-circularity, implying that the wind data's statistical properties may vary with rotation. For the medium-speed wind data, the compact cluster indicates less variability in wind speed, and the shape approaches circularity more than the high-speed regime. The low-speed wind data shows the smallest spread, indicating low variability and smaller wind speed magnitudes. Despite the visual appearance of near circularity, the ρ value, although smaller than the other regimes, is still greater than zero, confirming non-circularity.

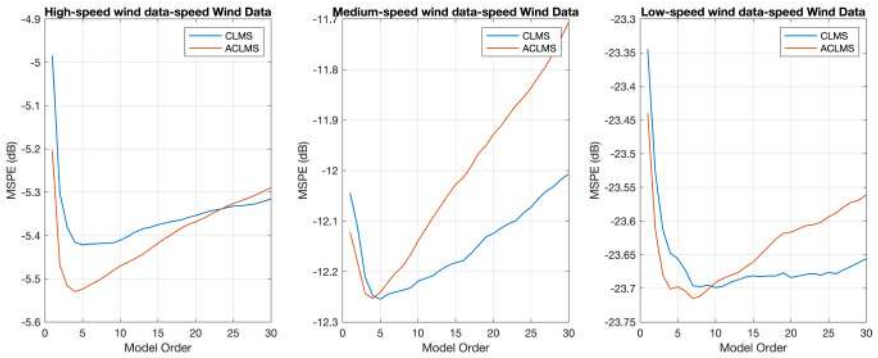

Figure 36: MSPE against filter order for 3 wind speed regimes for CLMS and ACLMS algorithms.

The CLMS and ACLMS algorithms were implemented to perform one step ahead prediction for three of the wind speed regimes. The optimal learning rate μ was chosen to be 0.001, 0.01, 0.1 for high, medium and low wind speed regimes respectively. These values allowed sufficient convergence of algorithms.

In the context of predicting complex wind data, the CLMS algorithm assumes the data is circular (rotationally invariant), while the ACLMS algorithm accounts for non-circularity by utilising both the signal and its complex conjugate. The performance of these algorithms varies depending on the statistical characteristics of the wind regime being modeled.

The experimental results across different filter lengths indicate the following:

For the high-speed wind regime, the ACLMS algorithm demonstrates a consistent improvement in prediction accuracy over CLMS. As the filter length increases, ACLMS benefits from the additional degrees of freedom, allowing it to capture the complex structure in the data more effectively. This is evident from the lower MSPE values achieved by ACLMS across nearly all model orders tested.

In the medium-speed wind regime, both algorithms show a marked decrease in MSPE at lower model orders. However, ACLMS again outperforms CLMS, particularly in the mid-range of model orders. This suggests that the medium-speed wind data has non-circular characteristics that the ACLMS algorithm can exploit to enhance prediction accuracy.

For the low-speed wind regime, ACLMS maintains a performance advantage over CLMS, especially as the model order increases beyond the initial dip in MSPE. This indicates that the low-speed wind data may exhibit a higher degree of non-circularity or have a more significant noise component, which ACLMS can model more accurately than CLMS.

3.1.2 (c) Frequency Estimation in Three Phase Power Systems - Circularity Diagrams of Complex Voltages

Estimating the nominal frequency within a three-phase power system by generating scenarios of both balanced and unbalanced conditions was performed. Utilizing the Clarke transformation, the system was projected onto two orthogonal axes, forming the $v_\alpha(n)$ and $v_\beta(n)$ components. This projection allowed for the establishment of a complex Clarke $(\alpha - \beta)$ voltage $v(n) = v_\alpha(n) + jv_\beta(n)$.

The balanced condition was achieved by ensuring uniform peak voltages across the phases, denoted as $Va(n) = Vb(n) = Vc(n) = V$, and by securing a consistent phase shift of 120 degrees between them, thus setting $\Delta b = \Delta c = 0$. Conversely, the unbalanced conditions were created by altering the magnitude and/or the phase of one or more phases within the system. The circulatory plots along with their circulatory coefficients for these conditions are presented in Fig.37.

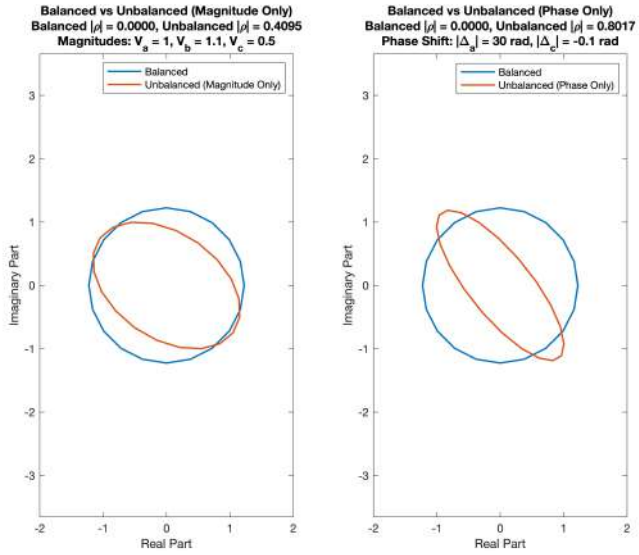

Figure 37: Circularity plots for unbalanced magnitude (Left) and Unbalanced phase (Right).

For a balanced system, the circularity diagram exhibits a perfect circle or a very closely approximated circular

shape.
In such conditions, the system operates optimally, with minimal phase distortion and equal load distribution across phases. The circularity coefficient in a balanced system, ideally, approaches zero, indicating a high degree of circularity In contrast, an unbalanced system deviates from this ideal circular shape in the circularity diagram.
Such conditions might be caused by asymmetric loads, faults in one or more phases, or inherent system irregularities. The circularity coefficient in an unbalanced system increases, signaling the departure from ideal circularity.
The circularity diagram functions as a pivotal diagnostic mechanism for identifying and scrutinizing faults within the system. By examining deviations from the circular configuration, engineers and system operators are equipped to deduce the existence and characteristics of system imbalances or faults. Specifically:

- **Magnitude Unbalance:** This distortion signifies a variance in the voltage magnitudes of the three phases. Any deviation from this equality points to an issue within the system, such as an overloaded circuit or a failing component, that necessitates further investigation and correction. The identification of which phase or phases are exhibiting variance can directly lead to pinpointing the fault's location and nature.
- **Phase Shift Distortion:** This could be the result of asymmetrical loads that disproportionately affect one or more phases or could stem from issues like improper wiring or damaged infrastructure. Such phase irregularities disrupt the harmonic balance of the system, potentially leading to inefficient operation or damage over time. Identifying phase shift distortions helps in diagnosing these underlying issues, enabling timely corrective measures.

3.1.3 (d) Derivation of the Frequency of Balanced and Unbalanced Voltages

Case 1 - Balanced Complex Voltage : Given the balanced system scenario, the complex Clarke $(\alpha - \beta)$ voltage $v(n)$ is represented as:

$$v(n) = \frac{\sqrt{3}}{2} V e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} \tag{45}$$

When incorporated into the strictly linear autoregressive model:

$$v(n+1) = h^*(n) v(n) \tag{46}$$

This yields the equation:

$$e^{j\left(2\pi \frac{f_0}{f_s}(n+1) + \phi\right)} = h^*(n) e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} \tag{47}$$

Given that $v(n+1)$ and $v(n)$ have identical magnitudes, the phase shift between consecutive samples, introduced by the factor $e^{j\left(2\pi \frac{f_0}{f_s}\right)}$, directly relates to the signal's frequency f_0 and the sampling frequency f_s. Expressing $h(n)$ in polar form yields:

$$h(n) = |h(n)| e^{-j\theta_h} \tag{48}$$

Accordingly, $h^*(n)$, the conjugate of $h(n)$, becomes:

$$h^*(n) = |h(n)| e^{j\theta_h} \tag{49}$$

The phase θ_h introduced by $h(n)$ correlates to the phase shift per sample, which is pivotal for deducing the frequency $f_0(n)$. From the relationship, we derive:

$$e^{j\left(2\pi \frac{f_0}{f_s}\right)} = |h(n)| e^{j\theta_h} \tag{50}$$

Given that the magnitude of $h(n)$ must equal 1 to preserve the equation's integrity (since both sides are purely phase shifts), the frequency f_0 is determined by:

$$f_0(n) = \frac{f_s}{2\pi} \theta_h(n) \tag{51}$$

By decomposing $h(n)$ into its real and imaginary components, $\theta_h(n)$ can be expressed as:

$$\theta_h(n) = \arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \tag{52}$$

Therefore, the formula for frequency becomes:

$$f_0(n) = \frac{f_s}{2\pi} \arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \tag{53}$$

This derivation provides a comprehensive understanding of how the frequency f_0 of the balanced system is intricately related to the autoregressive coefficient $h(n)$ within the framework of the strictly linear model, highlighting the fundamental connection between signal phase progression over time and its frequency.

Case 2 - Unbalanced Complex Voltage : Consider the complex-valued voltage expression for an unbalanced system as obtained from the Clarke Transform:

$$v(n) = A(n)e^{j\left(2\pi\frac{f_0}{f_s}n+\phi\right)} + B(n)e^{-j\left(2\pi\frac{f_0}{f_s}n+\phi\right)}, \tag{54}$$

where $A(n)$ and $B(n)$ represent the system's phase and magnitude variations, detailed as:

$$A(n) = \sqrt{\frac{6}{6}}\left[Va(n) + Vb(n)e^{j\Delta_b} + Vc(n)e^{j\Delta_c}\right], \tag{55}$$

$$B(n) = \sqrt{\frac{6}{6}}\left[Va(n) + Vb(n)e^{-j(\Delta_b+\frac{2\pi}{3})} + Vc(n)e^{-j(\Delta_c-\frac{2\pi}{3})}\right]. \tag{56}$$

Incorporating this into the widely linear autoregressive model gives:

$$v(n+1) = h^*(n)v(n) + g^*(n)v^*(n), \tag{57}$$

leading to the evolution of voltage representation for unbalanced conditions as follows:

$$A(n+1)e^{j\left(2\pi\frac{f_0}{f_s}\right)} = h^*(n)A(n) + g^*(n)B^*(n), \tag{58}$$

$$B(n+1)e^{-j\left(2\pi\frac{f_0}{f_s}\right)} = h^*(n)B(n) + g^*(n)A^*(n). \tag{59}$$

Assuming the voltages $Va(n)$, $Vb(n)$, and $Vc(n)$ change minimally over time, we equate $A(n+1)$ to $A(n)$ and $B(n+1)$ to $B(n)$, simplifying to:

$$e^{j\left(2\pi\frac{f_0}{f_s}\right)} = h^*(n) + g^*(n)\frac{B^*(n)}{A(n)}, \tag{60}$$

$$e^{-j\left(2\pi\frac{f_0}{f_s}\right)} = h^*(n) + g^*(n)\frac{A^*(n)}{B(n)}. \tag{61}$$

By analyzing the phase shift over a single sampling period implied by $e^{j\left(2\pi\frac{f_0}{f_s}\right)}$, the frequency $f_0(n)$ can be derived. The manipulation involves recognizing the conjugate relationship between terms, leading to a quadratic equation in terms of $B^*(n)/A(n)$ and eventually to an explicit expression for $f_0(n)$ as a function of model coefficients $h^*(n)$ and $g^*(n)$.
The expression for $f_0(n)$ reflects the unbalanced system's nature through the coefficients:

$$f_0(n) = \frac{f_s}{2\pi}\arctan\left(\frac{\sqrt{\Im^2\{h(n)\} + 4|g(n)|^2 - \Im^2\{g(n)\}}}{\Re\{h(n)\}}\right), \tag{62}$$

which constitutes the complete derivation for frequency estimation in unbalanced systems.

3.1.4 (e) Using CLMS and ACLMS Algorithm to Estimate Frequency of Complex Clarke Voltages

Fig.38 shows the estimated frequency for both balanced and unbalanced complex voltage systems using CLMS and ACLMS algorithms. For this section the nominal frequency for both systems was set to 50Hz (UK).

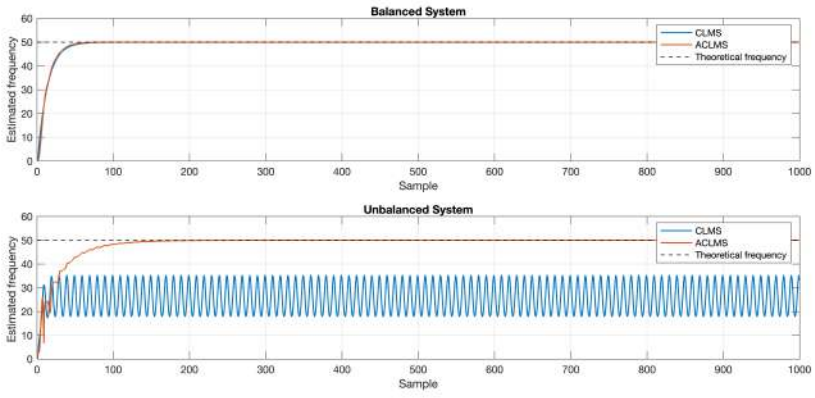

Figure 38: Estimated Frequency plots using CLMS and ACLMS algorithms for balanced and unbalanced complex voltage systems.

The CLMS algorithm is a linear adaptive filtering technique designed to minimize the error between a desired signal and its estimate by adjusting its coefficients in a complex domain. It's well-suited for stationary signals or

signals with properties that change slowly over time.
For a balanced system, where the voltages are symmetrical and phase shifts are exactly 120 degrees apart, the signal representation aligns well with the assumptions of the CLMS algorithm. In this scenario, the algorithm should accurately estimate the system's nominal frequency because the signal's complexity is reduced to a predictable pattern that CLMS can follow effectively. This is seen in Fig.38 (Top) as both CLMS and ACLMS quickly converge to the theoretical estimated frequency.

In contrast, unbalanced system voltages introduce asymmetry, deviating from the ideal conditions assumed by the CLMS algorithm. The presence of non-sinusoidal components and harmonic distortions can complicate the frequency estimation process. The CLMS algorithm, which relies on linear adaptation, might struggle to accurately capture and adapt to these complexities as seen in Fig.38 (Bottom) where it oscillates at an estimated frequency of around 39 Hz. In such cases, the Augmented CLMS (ACLMS) algorithm, which incorporates both the signal and its complex conjugate, might offer a more accurate estimation by better accounting for the additional complexities of unbalanced systems. The ACLMS algorithm's ability to adapt to both the linear and non-linear components of a signal makes it more suited for handling the irregularities present in unbalanced systems. This is seen in Fig.38 (Bottom) as it quickly adapts and converges to the theoretical estimated frequency of 50 Hz in less than 150 samples.

3.2 Adaptive AR Model Based Time-Frequency Estimation

3.2.1 (a) Finding the AR Coefficients of a FM Signal

A frequency modulated (FM) signal shown in Fig.39 (Left) with a length of 1500 samples is generated by the expression

$$y(n) = e^{j(2\pi\phi(n))} + \eta(n) \tag{63}$$

where $f_s = 1500\,\text{Hz}$ represents the sampling frequency. Here, $\eta(n)$ is circular complex white Gaussian noise with zero mean and a variance of 0.05. The phase $\phi(n)$ is determined by the cumulative sum of a time-dependent frequency function $f(n)$, reflecting the non-stationary frequency characteristic of the complex signal, defined as

$$\phi(n) = \int Rf(n)\,dn. \tag{64}$$

The AR(1) coefficients were then found for the complete signal and the power spectrum was plotted in Fig.39 (Middle).

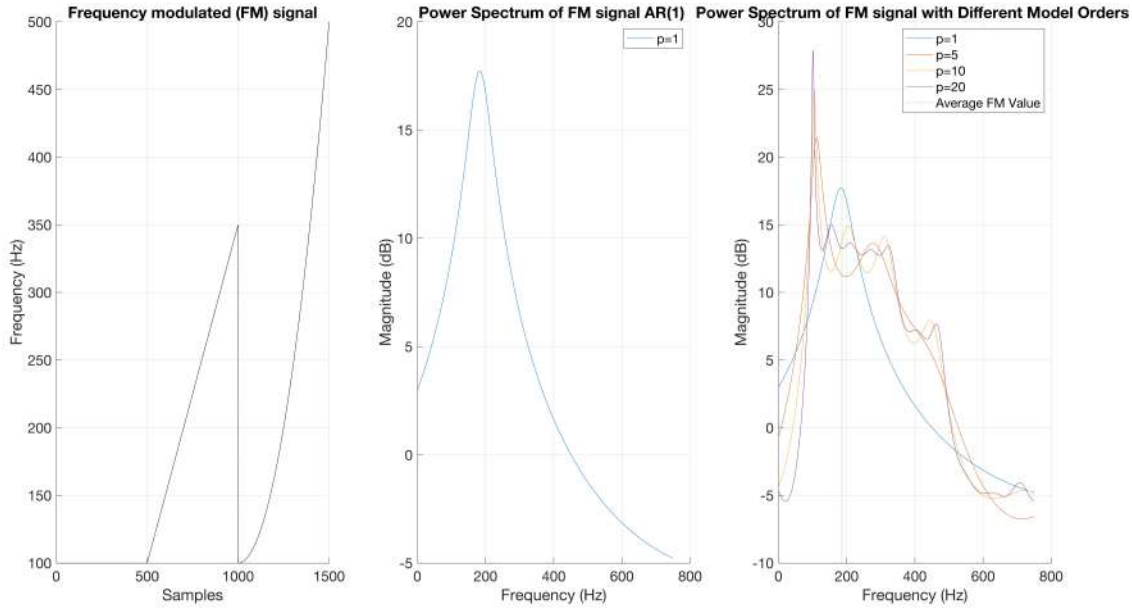

Figure 39: Time varying frequency of the FM signal (Left). Power Spectrum of the FM signal AR(1) Model (Middle). Power spectrum of the FM signal with different AR Models.

The `aryule` function is based on the premise that the analyzed signal is stationary. Consequently, the power spectrum estimation shown in Fig. 39 (Middle) exhibits a single peak at approximately 180.6 Hz. This peak corresponds to the average frequency of the signal over the entire time span. Due to the stationary assumption of the `aryule` function, the model does not account for the temporal variations of the signal's frequency, $f(n)$. The AR(1) model, therefore, oversimplifies the signal, which leads to an inadequate representation of the frequency modulation that varies with time.

As observed in Fig.39 (Right), a better AR model, even with a higher order, fails to accurately trace the frequency transitions within the complete signal. An increase in the model order leads to a greater number of spectral peaks. Notably, at a model order of 5 and 10, a distinct peak at around 100 Hz correlates with the stationary segment of $f(n)$. However, the AR model falls short in precisely capturing other frequency changes in the signal. Therefore, the AR models for the entire signal do not correctly estimate the changes in frequency.

3.2.2 (b) Implementing CLMS Algorithm to Find AR Coefficients of FM Signal

The CLMS Algorithms with three different learning rates was implemented on MATLAB to find the AR coefficients of the complete signal in section 3.2.1. The results are shown in Fig.40.

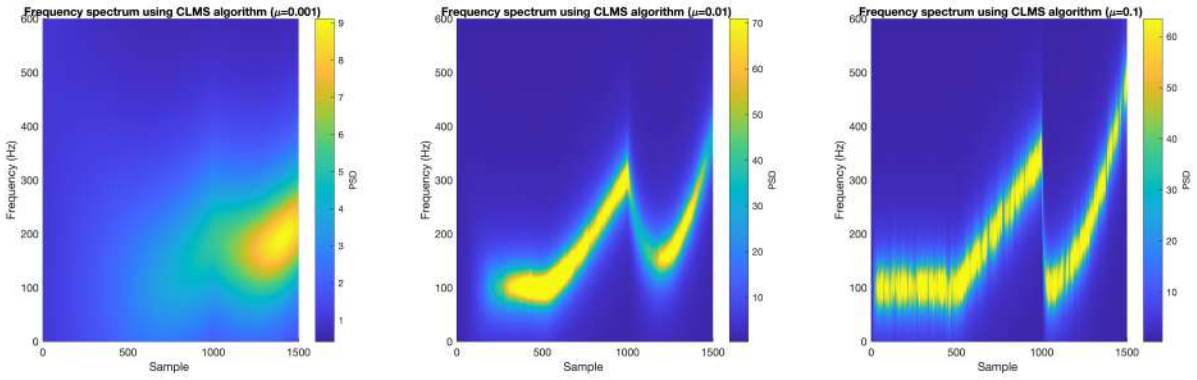

Figure 40: Time frequency spectrum of FM signal with 3 different learning rates using the CLMS algorithm.

Given that the data exhibits a high degree of circularity, the CLMS (Complex Least Mean Square) algorithm is particularly well-suited for modeling the signal process in the form of $y(n) = a_1^* y(n-1)$. This modeling approach allows for the adaptive and iterative update of the coefficient estimate $\hat{a}_1^*$, enabling the algorithm to effectively track and adapt to the non-stationary frequency changes within the data.

Unlike traditional AR coefficient estimation methods such as `aryule`, which compute a block estimate of the AR coefficient for the entire signal at once, the CLMS algorithm updates the coefficient estimate incrementally. This iterative process relies solely on the immediate past samples of the signal $y(n)$, rather than the complete dataset. As a result, the CLMS algorithm can dynamically adjust the coefficient in response to ongoing changes within the signal, enhancing its ability to model non-stationary behavior effectively.

The effect on learning rates of the CLMS algorithm is also observed in Fig.40. For low values of μ (0.001), the convergence of the CLMS algorithm is notably slow. Consequently, the AR coefficient updates inadequately capture the variations in f(n), leading to spectral estimates that fail to reach the true values.
Conversely, higher values of μ (0.1) introduce noise into the spectrum, causing the frequency estimates to oscillate around the true value giving a less stable estimate.

Thus, the choice of an appropriate learning rate is crucial in achieving accurate and stable spectral estimates, ensuring an effective balance between convergence speed and noise resilience.

3.3 A Real Time Spectrum Analyser Using Least Mean Square

3.3.1 (a) Obtaining the Least Squares Solution

To estimate a signal $y(n)$ using a linear combination of N harmonically related sinusoids, we start with the equation:

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k) e^{j2\pi kn/N} \tag{65}$$

This can be expressed in vector form as:

$$\hat{y} = Fw \tag{66}$$

where F represents the Discrete Fourier Transform (DFT) matrix. The optimal weights w are determined by minimizing the sum of squared errors between the estimated signal and the true signal:

$$\min ||y - \hat{y}||^2 = \min ||y - Fw||^2 \tag{67}$$

Expanding this expression, we obtain:

$$\min ||y - \hat{y}||^2 = \min (y - Fw)^H (y - Fw) \tag{68}$$

$$= \min(y^H y - w^H F^H y - y^H F w + w^H F^H F w)$$

After simplification, we arrive at:

$$= \min(y^H y - 2w^H F^H y + w^H F^H F w) \tag{69}$$

To find the minimum, we take the gradient with respect to w:

$$\nabla_w ||y - \hat{y}||^2 = -2F^H y + 2F^H F w \tag{70}$$

Setting this gradient to zero, we find the equation for the optimal coefficients in the least-squares sense:

$$-2F^H y + 2F^H F w = 0 \tag{71}$$

This simplifies to:

$$F^H F w = F^H y \tag{72}$$

Finally, solving for w, we obtain:

$$w = (F^H F)^{-1} F^H y \tag{73}$$

This equation gives the optimal coefficients w that minimize the sum of squared errors, providing the least squares solution for estimating the signal $y(n)$. Comparing this with the inverse Discrete Fourier Transform (IDFT) equation, which computes the signal from its frequency-domain representation:

$$y = F^H w \tag{74}$$

It is observed that the least squares solution for estimating $y(n)$ is similar to the IDFT equation. Both involve the operation of F^H, the conjugate transpose of the DFT matrix F, on the coefficient vector w. However, in the least squares solution, we additionally multiply by $(F^H F)^{-1}$, which accounts for the least squares minimization of the error. Nonetheless, the fundamental operation remains the same - transforming the coefficient vector from the frequency domain to the time domain using the conjugate transpose of the DFT matrix.

3.3.2 (b) Fourier transform in terms of the change of basis and projections.

In part (a), we derived the least squares solution for estimating a signal $y(n)$ using a linear combination of harmonically related sinusoids, represented by the equation $\hat{y} = Fw$, where F is the Discrete Fourier Transform (DFT) matrix and w are the optimal weights.

Now, let's consider the geometric interpretation of this least squares solution. Since $\hat{y} = Fw$, the estimate $\hat{y}$ lies within the column space of F, denoted as $C(F)$, which is spanned by the basis vectors formed by the columns of F. Therefore, $\hat{y}$ can be seen as the projection of y onto $C(F)$ that minimizes the sum of squared errors.

This interpretation is further reinforced by the use of the projection matrix P, which projects a vector onto $C(F)$. The projection matrix P is obtained by substituting the least squares solution $w = (F^H F)^{-1} F^H y$ into the expression for $\hat{y}$, resulting in:

$$P = F(F^H F)^{-1} F^H y = Py \tag{75}$$

This equation emphasizes that the estimate $\hat{y}$ is indeed the projection of y onto the column space $C(F)$ using the projection matrix P, thereby minimizing the sum of squared errors.
Therefore, both the algebraic derivation in part (a) and the geometric interpretation highlight the least squares solution's role in finding the projection of the signal onto the column space of the DFT matrix, ultimately providing an optimal estimate of the signal in terms of minimizing squared errors.

3.3.3 (c) Implementing DFT-CLMS Algorithm on FM Signal

Treating the DFT as a least squares solution allows for the implementation of an adaptive version using the CLMS algorithm. This technique, known as the DFT-CLMS algorithm, facilitates the iterative updating of Fourier coefficients in a manner similar to online adaptive filtering. At each time instant, the input to the CLMS algorithm is the DFT basis function corresponding to the current time index. Consequently, the magnitude of the weight vector of the adaptive filter obtained through the CLMS algorithm provides an estimate of the frequency spectrum at every time instant.

This approach offers a dynamic and adaptive method for spectral estimation, where the frequency content of the signal can be tracked in real-time. By updating the Fourier coefficients iteratively, the DFT-CLMS algorithm can adapt to changes in the signal's spectral characteristics over time. Additionally, the use of the CLMS algorithm allows for efficient implementation and computational scalability, making it suitable for real-time applications where the signal's frequency content may vary or evolve dynamically.

The DFT-CLMS Algorithm was implemented on the FM Signal and the results are plotted on Fig.41.

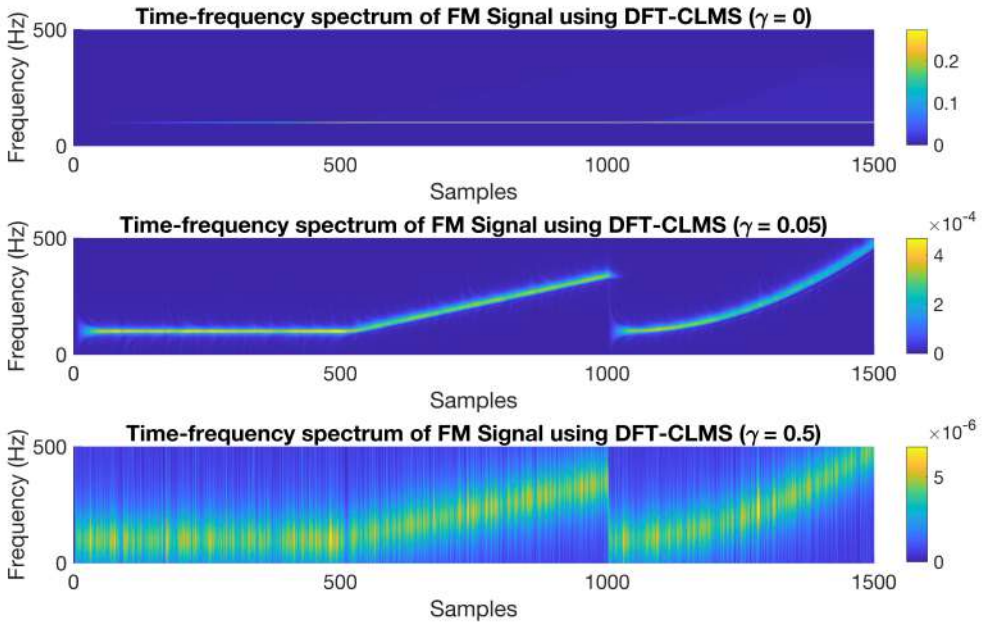

Figure 41: Time frequency spectrum of FM signal with 3 different γ using the DFT-CLMS algorithm.

The AR-spectrum analyser in Fig.40 outperforms the DFT-CLMS in seen in Fig.41 for the non-stationary parts of the signal.The Discrete Fourier Transform assumes that the signal is periodic within the analyzed window and therefore does not capture transient or non-stationary components well. Additionally, the DFT is sensitive to the window size and may not accurately represent the frequency content of signals with rapidly changing spectra.

By observing Fig.40 (Middle), we understand that increasing the leaky coefficient can be a way of improving the DFT-CLMS. As the value of γ increases, the leakage term becomes more dominant in the weight update equation. This helps stabilize the adaptation process by reducing the impact of transient fluctuations in the input signal. Consequently, the algorithm converges more smoothly and efficiently, leading to improved spectral estimation performance. In signals with non-stationary or time-varying spectral characteristics, a higher leakage parameter can help the DFT-CLMS algorithm adapt more effectively to changes in the signal's frequency content. By providing smoother and more stable updates to the Fourier coefficients, the leaky DFT-CLMS algorithm can more accurately track variations in the signal's spectrum over time.

Overall, while increasing the leakage parameter can provide benefits such as improved stability and enhanced frequency resolution, it is essential to strike a balance and avoid excessive regularization. Beyond a certain threshold, higher γ values can lead to detrimental effects such as excessive smoothing, reduced adaptation speed, increased bias, and loss of sensitivity, ultimately compromising the accuracy and reliability of the spectral estimation process. This can be seen in Fig.40 (Bottom).

3.3.4 (d) Implementing DFT-CLMS Algorithm on EEG Signal

The DFT-CLMS Algorithm from section 3.3.3 was implemnted on EEG (POz) data. Since the data was stationary a leaky DFT-CLMS was not used and the γ value was set to 0. The time frequency spectrum for the selected EEG data segment is shown in Fig.42.

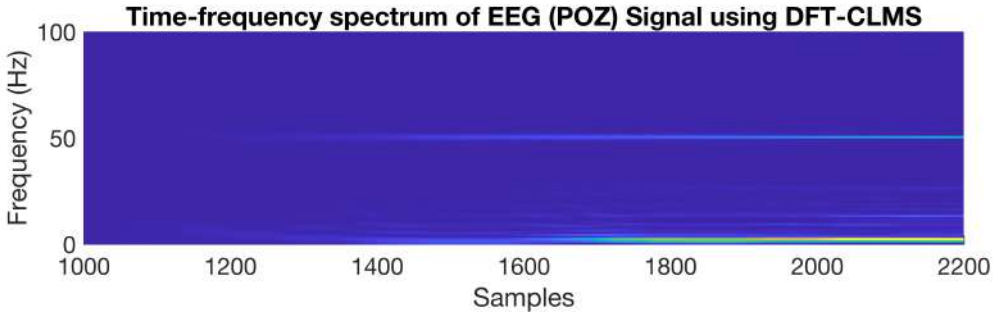

Figure 42: Time frequency spectrum of FM signal with 3 different γ using the DFT-CLMS algorithm.

From the spectrum we can clearly observe the 50 Hz power line interference as well as the first harmonic of SSVEP at 13Hz. However, higher harmonics are not clearly visible. The DFT-CLMS algorithm provides a high-resolution representation of the frequency content of the EEG signal. By estimating the spectral components using the Discrete Fourier Transform (DFT) basis functions, it can accurately capture subtle variations in the signal's frequency spectrum, making it suitable for analyzing stationary EEG signals.

4 From LMS to Deep Learning

4.1 Neural Networks for Prediction

4.1.1 1. One Time Step Ahead Prediction using Linear LMS

Prior to applying LMS algorithm on the time series data provided, the data is pre-processed to remove the mean. The original signal and the zero-mean signal are shown in Fig.43.

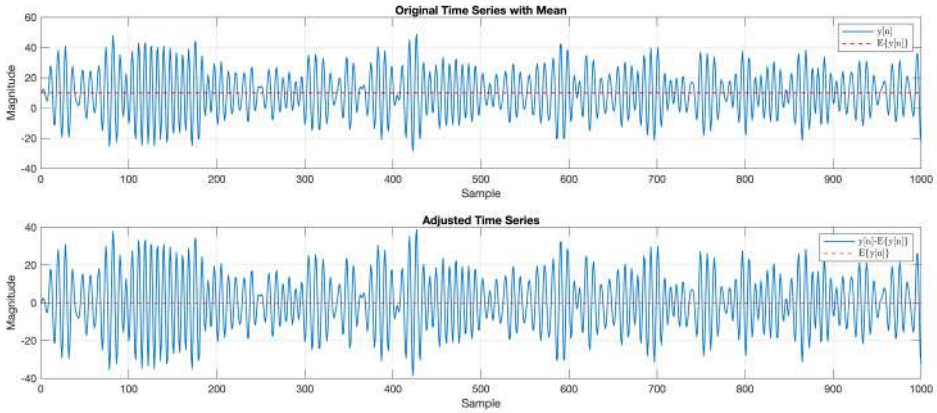

Figure 43: Original Time series (Top) and Zero-mean time series (Bottom)

The LMS algorithm ($\mu = \times 10^{-5}$) from Section 3 is applied to perform a one step ahead prediction of the zero-mean signal modelled as an AR(4) process. The plot of the zero-mean signal against its one-step prediction is plotted in Fig.44.

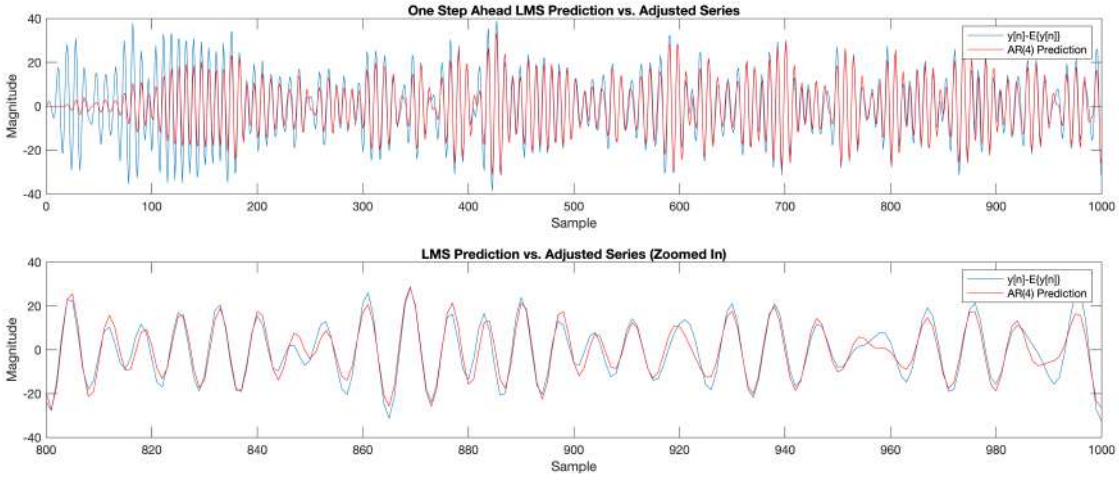

Figure 44: Linear LMS one time step ahead prediction of time series data.

From Fig.44, the behavior of the output signal from the LMS algorithm reflects a convergence to the characteristics of the original zero-mean time series after approximately 200 iterations. This trend indicates that the LMS algorithm is successfully adapting its coefficients to the underlying pattern of the data.
There are regions where the prediction does not match perfectly, which is expected since LMS provides an estimate based on the previous values and cannot account for random fluctuations due to noise perfectly.

Quantitatively, the mean-square error (MSE) is calculated to be 16.03 dB. The MSE is a critical metric that signifies the accuracy of the predictions—the lower the MSE, the closer the predictions are to the true values. An MSE of

16.03 dB suggests that there are errors present, which are expected given the non-stationary nature of the data and the prediction methodology.

Additionally, the prediction gain R is calculated to be 5.196 dB. This metric offers insights into the noise reduction achieved by the LMS output relative to the variance of the prediction error. The prediction gain is computed from the ratio of the variance of the LMS output, denoted as σ^2, to the variance of the prediction error, denoted as σ_e^2, with both quantities expressed in decibels. A prediction gain of 5.196 dB suggests that the LMS algorithm's predictions, while not perfect, provide a substantial improvement over a model with no predictive ability.

4.1.2 2. Using Non-Linear LMS Algorithm for Prediction of Time Series

When dealing with non-linear data, the linear model assumed by the basic LMS algorithm in section 4.1.1 may not be sufficient to capture the underlying patterns. Adding a non-linearity, such as an activation function, allows the model to accommodate more complex data structures. In the case of the time series prediction, incorporating an activation function like the hyperbolic tangent (tanh) can enhance the model's expressiveness by enabling it to model non-linear relationships within the data.

Fig.45 shows the dynamical perceptron prediction output of the zero-mean time series signal.

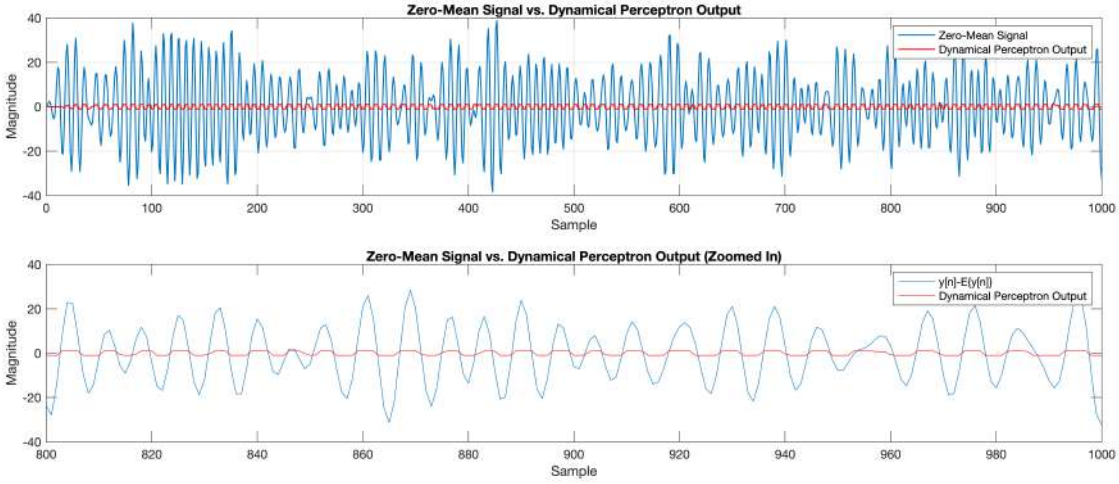

Figure 45: Zero-mean signal against its one-step ahead prediction using non-linear LMS with tanh activation function.

In the enhanced model, the predicted signal at time step n is given by

$$\hat{y}(n) = \tanh(\mathbf{w}^{\mathsf{T}}\mathbf{x}), \tag{76}$$

where $\mathbf{w}$ is the coefficient vector of the non-linear AR(4) model, and $\mathbf{x}$ encapsulates the four previous signal values of $y(n)$. The model employs an iterative refinement approach via a gradient-descent-based learning rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\left[1 - \tanh^2\left(\mathbf{w}^{\mathsf{T}}(n)\mathbf{x}(n)\right)\right]\mathbf{x}(n) \tag{77}$$

Here, μ denotes the learning rate, and $e(n)$ is the prediction error at the n-th step. This updating mechanism aims to minimize the cost function defined by:

$$J(n) = \frac{1}{2}e^2(n) = \frac{1}{2}\left[y(n) - \tanh\left(\mathbf{w}^{\mathsf{T}}(n)\mathbf{x}(n)\right)\right]^2 \tag{78}$$

For the dynamical perceptron output the MSE was found to be 22.96 dB and the prediction gain was -25 dB. This suggests that the predictions made by the model diverge significantly from the actual data. This reasoning is also supported by Fig.45.

The tanh function, a popular choice for introducing non-linearity in neural networks, maps input values to a range between -1 and 1. This characteristic can be particularly useful for certain types of data and predictions.
If the signal or the residuals of the model do not naturally fit within the -1 to 1 range, the tanh function might compress the predictions in a way that does not reflect the true variability or dynamics of the underlying data. This is evident in Fig.45, where the output signal is clipped.
The significant negative prediction gain also suggests that the model, with the tanh activation function, might be amplifying the noise rather than extracting useful signal from the data. This could indicate that the tanh function's properties are not aligning well with the signal's characteristics, leading to poorer model performance.
These results suggest that either a different activation function should be considered—one that offers a better fit for the data's characteristics.

4.1.3 3. Scaled Non-Linear LMS Prediction

Generalizing the activation function by scaling its amplitude, as in $a \cdot \tanh$, allows you to adjust the function's output range from the standard $[-1, 1]$ to $[-a, a]$. This adjustment can make the activation function more flexible, potentially enabling it to better fit the specific characteristics of the data, by preventing output clipping.

The necessity for a to exceed the maximum absolute value of the zero-mean signal, thereby ensuring $a \geq 39$. This is crucial to prevent the clipping of the activation function's output, allowing the model to fully capture the range of the time series data. To refine a and w effectively over time, we employ gradient descent, leveraging the error signal, $e(n)$, and the input vector, $x(n)$, at each timestep n. The updates are designed to minimize prediction error by adjusting a and w iteratively, based on the gradient of the error with respect to these parameters.

The update rule for the scaling factor, a, can be described as follows:

$$a = a + \mu e(n) \tanh(\mathbf{w}(n)^{\mathsf{T}} \mathbf{x}(n)) \tag{79}$$

Simultaneously, the weights are updated as per:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu a e(n) \left(1 - \tanh^2(\mathbf{w}(n)^{\mathsf{T}} \mathbf{x}(n))\right) \mathbf{x}(n) \tag{80}$$

The optimal value of a is found to be 92.5 for a constant learning rate of $\mu = 1 \times 10^{-7}$. This scaling value gives us the lowest MSE (6.67 dB) and the highest prediction gain (16.6 dB). The plot of the output of LMS algorithm with scaled tanh function is shown in Fig.46.

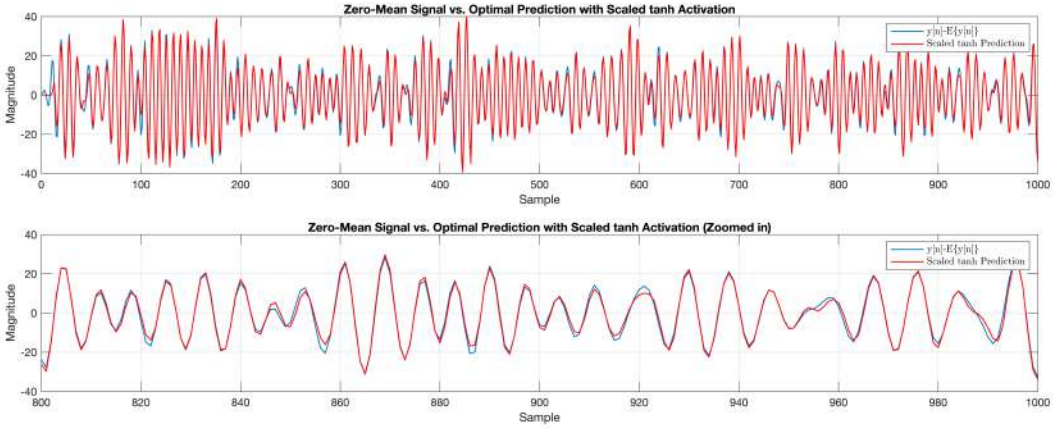

Figure 46: Zero-mean signal against its one-step ahead prediction using non-linear LMS with scaled tanh activation function.

By comparing the plot in Fig.46 as well as the quantitative values of MSE and prediction gain with the ones in section 4.1.1 and 4.1.2, we conclude that scaled non-linear LMS algorithm outperforms the linear LMS as well the non-scaled non-linear LMS.

4.1.4 4. Non-Linear LMS Prediction with Bias

Adding a bias term to a model, such as a neural network or any machine learning model utilizing gradient descent for optimization, is a standard technique to make the model more flexible and capable of fitting data with a non-zero mean more effectively. The bias term allows the activation function's output to shift accordingly, enabling the model to better capture the underlying data distribution, including its mean.

In this refined approach, the model's prediction, denoted as $\hat{y}(n)$, integrates a bias term b, in addition to employing a scaled hyperbolic tangent activation function. The prediction at each time step is mathematically represented as:

$$\hat{y}(n) = a \cdot \tanh(\mathbf{w}^{\mathsf{T}}(n)\mathbf{x}(n) + b), \tag{81}$$

where a signifies the scaling factor, $\mathbf{w}(n)$ corresponds to the weights at time n, and $\mathbf{x}(n)$ encapsulates the input, specifically the preceding four values of the time series $y(n)$. To seamlessly incorporate the bias within the model's framework, the input is augmented by prefixing a constant 1, thereby extending the dimensionality of the coefficient array to encompass the bias term. Consequently, the bias at each step, $b(n)$, is effectively extracted as the initial weight $w_0(n)$, ensuring the model's adjustment to the non-zero mean characteristic of the data.

Fig.47 shows the prediction output of the scaled and biased non-linear LMS algorithm.

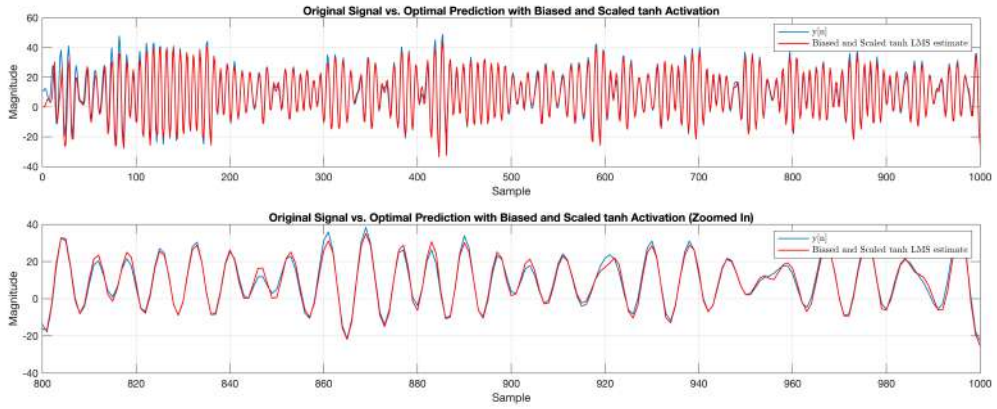

Figure 47: Original signal against its one-step ahead prediction using non-linear LMS with biased and scaled tanh activation function.

The performance of this algorithm falls short when compared to predictions made on zero-mean signals in previous sections. This is primarily because errors are more prevalent in the initial phase, prior to learning the bias, as indicated by an MSE of 15.44 dB and a prediction gain of 10.70 dB. Nonetheless, after reaching convergence, the algorithm's effectiveness aligns closely with that of predictions made ON zero-mean signal, as indicated by an MSE of 6.28 dB and a prediction gain of 16.6 dB.

4.1.5 5. LMS Prediction by Pre-Training Weights

Using the LMS algorithm ($\mu = 10^7$ and $a = 68$) with weights that have been pre-trained through overfitting to a limited subset of signal samples—specifically, the first 20 samples across 100 iterations, facilitates a significant enhancement in the algorithm's overall performance.

Fig.48 shows the prediction output of the pre-trained and non-linear LMS algorithm.

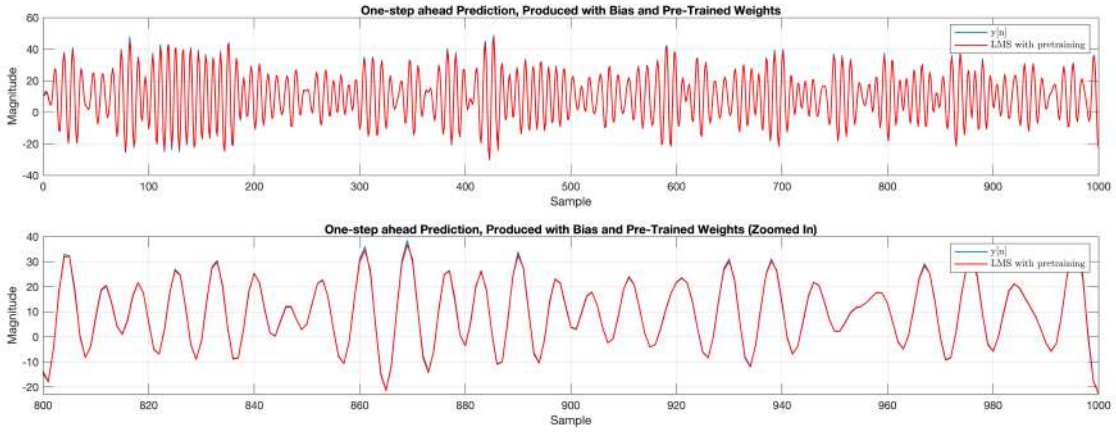

Figure 48: Original signal against its one-step ahead prediction using non-linear LMS with bias and pre-training.

From Fig.48 we observe that the LMS algorithm with pre-trianing and bias has the fastest convergence rate compared to other algorithms employed in Section 4.
Moreover, the estimated output very closely matches the input signal. Pre-training the weights on a small, focused subset of the time-series data allows the algorithm to "learn" important signal characteristics early on. This head start significantly reduces the time required for the algorithm to adjust to the broader dataset, resulting in quicker convergence.
Quantitatively, the MSE for this method is found to be 6.38 dB and the prediction gain is 16.98 dB which is an improvement in both the values when compared to the MSE and prediction gain obtained in Section 4.1.4.

4.1.6 6. Training a Deep Network on Backpropogation Algorithm

In Figure 11 given in Coursework document, we observe a deep neural network with three hidden layers, designed for the prediction of non-stationary time-series data. This kind of deep architecture is capable of modeling the complex temporal dynamics inherent in time-series which are not adequately captured by simpler, shallower models.

The network processes a sequence of inputs $x[n-1], x[n-2], x[n-3], x[n-4]$, each corresponding to a time-series observation at a consecutive time step. Each input node is connected to the subsequent layer of nodes (neurons), and these connections represent the weights that the backpropagation algorithm will learn to adjust.

Utilizing the network depicted in Figure 11, backpropagation proceeds as follows:

- Forward Pass:
  The input series $x[n-4], \ldots, x[n-1]$ is propagated through the network. Each layer computes a transformation through weighted sums and activation functions (Hidden Layer) to produce the output $\hat{x}[n]$.
- Loss Computation:
  The discrepancy between the predicted output $\hat{x}[n]$ and the actual value $x[n]$ is quantified using the loss function, commonly the Mean Squared Error (MSE):

$$\mathcal{L} = \frac{1}{2}(x[n] - \hat{x}[n])^2 \tag{82}$$

- Backward Propagation:
  The loss gradient with respect to the weights is computed by applying the chain rule and is then backpropagated through the network:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \hat{x}[n]} \cdot \frac{\partial \hat{x}[n]}{\partial w} \tag{83}$$

- Weight Update:
  The weights are adjusted in the direction that reduces the error, scaled by the learning rate η:

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial \mathcal{L}}{\partial w} \tag{84}$$

  Through iterative refinement by repeating these steps, the network learns to minimize the loss, enabling the modeling of complex time-series data. The depth of the network plays a pivotal role in capturing subtle patterns and high-level features beyond the reach of simpler models or shallower networks.

4.1.7 7. Training a Deep Network

A time-series exhibiting non-linear characteristics was created by superposing ten sinusoidal functions, each with distinct frequencies, phases, and amplitudes. To this signal, noise was introduced at a noise power 0.05, resulting in the composite series depicted in Fig.49.

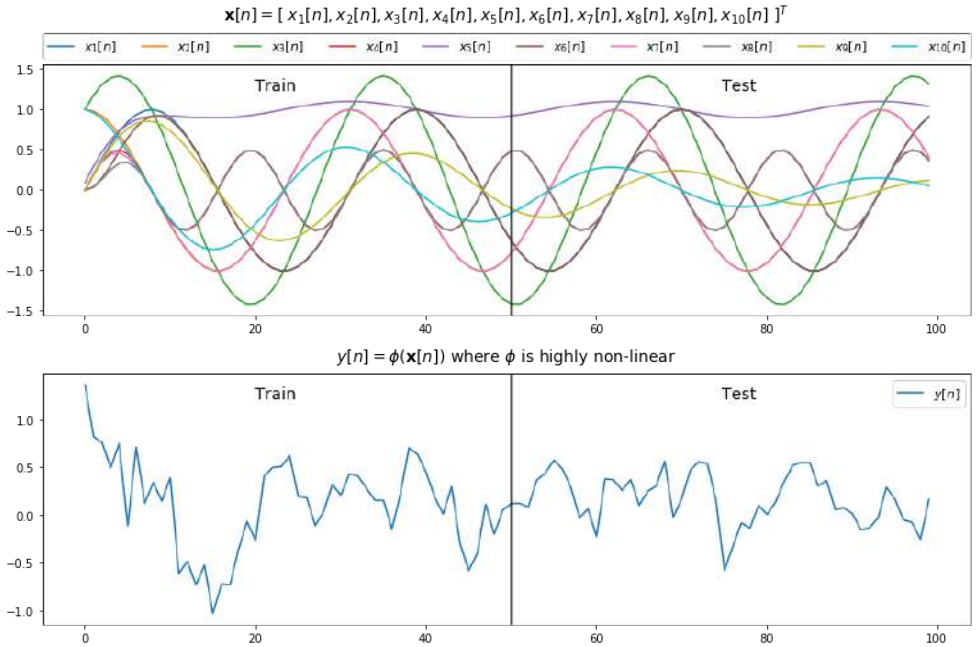

Figure 49: A time-series exhibiting non-linear characteristics made up of sinusoids (Top). Noise corrupted output signal (Bottom).

In the neural network's framework, the terms $x_1[n]$, $x_2[n]$, and similar variables represent the elements of the input vector at each timestep n. These inputs serve as the features that the neural network analyzes to derive predictions.

The process involves the transformation of these inputs through the network's architecture, ultimately yielding $y[n]$ as the estimated output signal. The output $y[n]$ encapsulates the neural network's prediction or decision based on the input features it received.

Three predictive models were used: a linear LMS algorithm, a dynamical perceptron with a tanh activation function, and a complex deep learning neural network with ReLU activation function. The deep network was trained over 20,000 epochs with four hidden layers having 10, 5, 5, and 5 neurons each, ending with a single output neuron. The training used a learning rate of 0.01.

The output prediction from the three models are shown in Fig.50. The performance is compared both on the training set (left side) and the test set (right side).

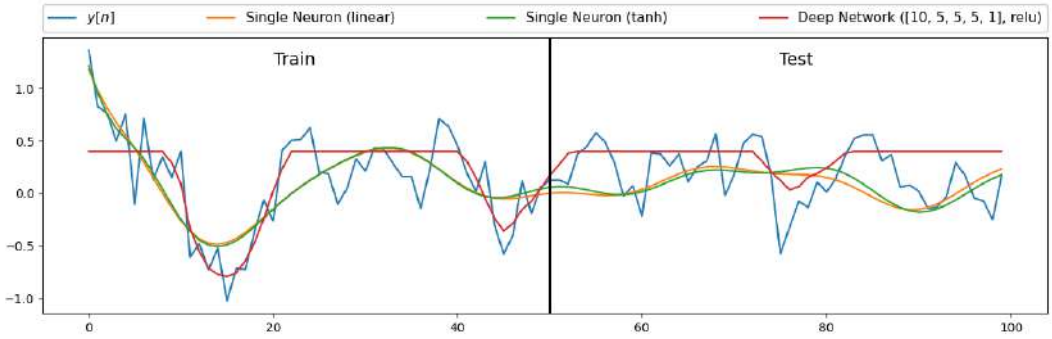

Figure 50: Linear LMS, dynamical perceptron with tanh and deep network output predictions of non-linear time-series

In the provided output prediction plot of Fig.50, the simple models with linear and tanh activations can capture the overall trend of the data but fail to match the finer details. The linear model's limitations are expected due to its simplicity, while the tanh model behaves similarly because it effectively acts as a linear function for a large portion of its inputs. This function acts nearly linearly within the -0.5 to 0.5 input range. Both models lack the complexity needed to fully mimic the intricate variations of the signal.
The deep network, despite its more sophisticated architecture, seems to smooth out the signal, resulting in a loss of finer detail and an underrepresentation of the signal's amplitude variations. This smoothing effect suggests that the network might be averaging over the data points, which could be due to the ReLU activation functions in the hidden layers or the way the network's capacity has been utilized.

The loss curves as a function of epochs for the linear LMS, non-linear perceptron, and deep network reveal distinct learning behaviors for each model. The results are shown in Fig.51.

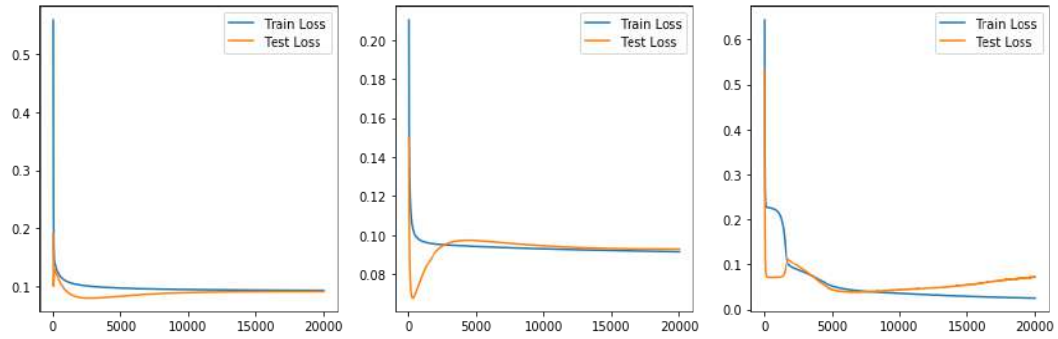

Figure 51: Loss curves as a function of epoch count for Linear LMS (Left), dynamical perceptron with tanh (Middle) and deep network (Right)

For the linear LMS (Left), both the training and test loss decrease rapidly at first and then plateau, indicating that the model quickly reaches its performance limit. This is expected as the linear LMS is limited in its ability to model complex, non-linear data.
The non-linear perceptron (Middle) shows a more gradual decline in loss, which is consistent with its ability to capture some non-linear relationships in the data due to the tanh activation function. However, the test loss stops improving at a certain point, suggesting a limitation in its capacity to fully grasp the time-series complexity.
The deep network (Right) presents a markedly different loss trajectory. It achieves a significant reduction in training loss, reflecting its capability to fit the training data closely. Yet, the test loss initially decreases before flattening out,

indicating good generalization performance but with signs of overfitting, as evidenced by the gap between the training and test loss.

4.1.8 8. Effect of Noise Power on Output Prediction of Deep Networks

Deep learning models, especially those with multiple layers, have high representational power, allowing them to capture complex patterns in data. However, this capability can also lead them to overfit, particularly when noise is present in the training data. Overfitting means the model learns the noise as if it were a meaningful part of the signal, which degrades its performance on unseen, test data.
The deep network from Section 4.1.7 is trained with different noise powers of 0 and 0.5.

Fig.52 shows the prediction output from the 3 models using a noise power of 0 and Fig. 53 shows the loss curves as a function of epochs for the three models with noise power of 0.

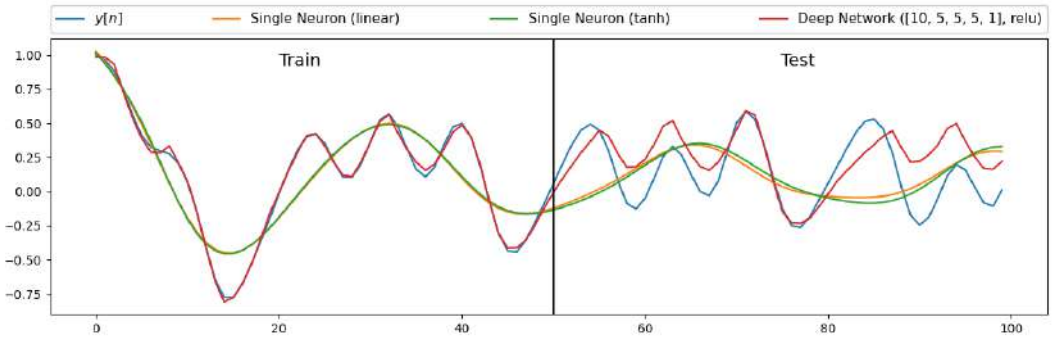

Figure 52: Linear LMS, dynamical perceptron with tanh and deep network output predictions of non-linear time-series using noise power of 0.

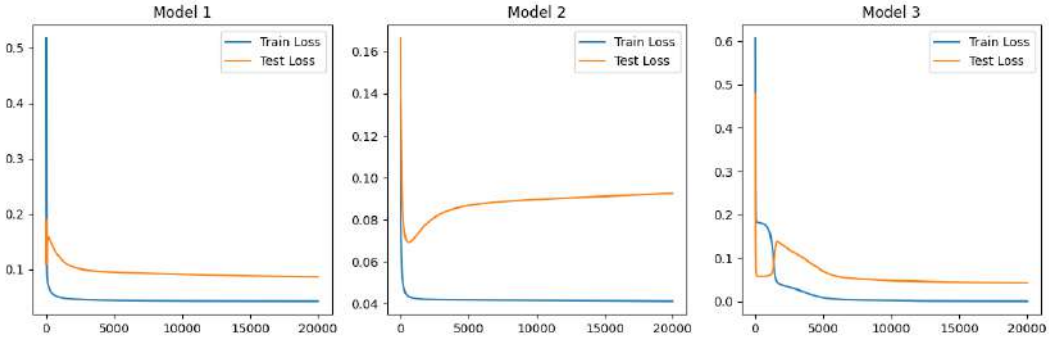

Figure 53: Loss curves as a function of epoch count for Linear LMS (Left), dynamical perceptron with tanh (Middle) and deep network (Right) for noise power of 0.

The three models—Linear LMS, Tanh perceptron, and Deep Network—demonstrate distinct behaviors in a noise-free environment. The Linear LMS model rapidly converges to a stable loss, reflecting its simplicity and limited capacity, as it can only capture the general trend but not the detailed fluctuations of the signal.This model captures the linear aspects of the data reasonably well, but since there's no noise, its limitations become clear—it cannot capture any non-linear trends in the signal as seen in Fig.52. The Tanh perceptron exhibits a more gradual learning curve, suggesting some capacity to account for non-linearities, yet it also plateaus, indicating a bounded ability to model the signal's complexity. The Deep Network shows a steep reduction in loss, highlighting its potential for intricate pattern recognition, though a slight rise in test loss suggests minor overfitting.

Comparatively, the Deep Network aligns more closely with the true signal in the prediction plot, affirming its superior capacity over the simpler models. The loss curves corroborate this by displaying the lowest final loss values, despite the early hints of overfitting. Overall, in the absence of noise, the Deep Network's architecture allows for a more nuanced signal approximation, but careful tuning is essential to fully harness its capabilities without overfitting.

Fig.54 shows the prediction output from the 3 models using a noise power of 0.5 and Fig.55 shows the loss curves as a function of epochs for the three models with noise power of 0.5.

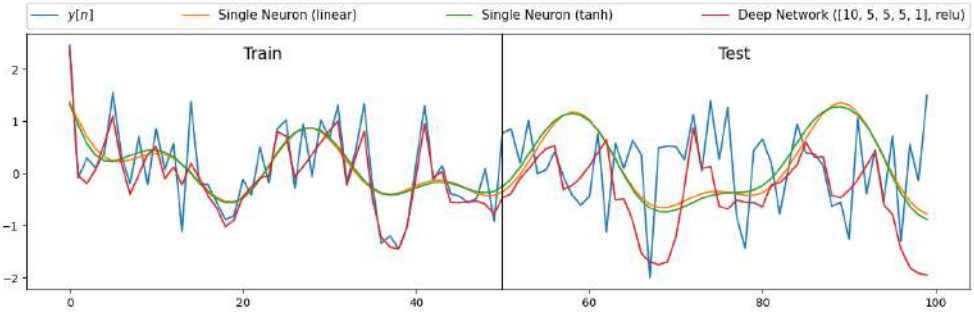

Figure 54: Linear LMS, dynamical perceptron with tanh and deep network output predictions of non-linear time-series using noise power of 0.5.

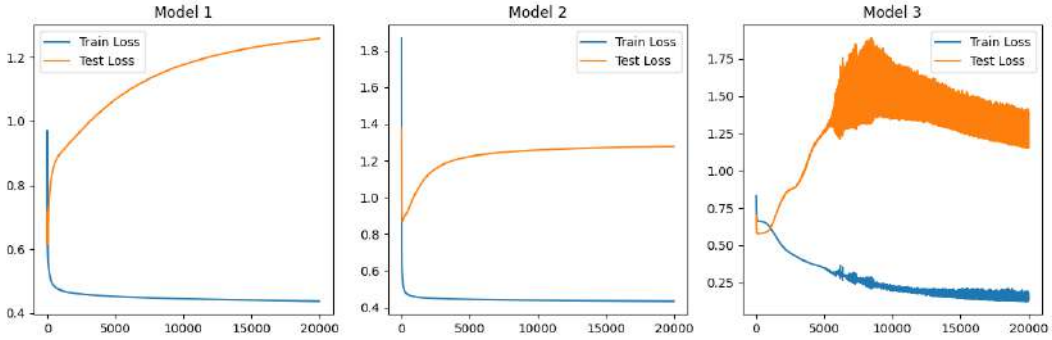

Figure 55: Loss curves as a function of epoch count for Linear LMS (Left), dynamical perceptron with tanh (Middle) and deep network (Right) for noise power of 0.5.

In an environment with a noise power of 0.5, the observed performance of the three models displays a marked difference from the noise-free scenario. The Linear LMS model is significantly challenged by the noise, with its predictions diverging notably from the true signal, which is also reflected in a high and stable test loss, indicating a lack of capacity to model the noisy data accurately. The Tanh Perceptron, while benefiting slightly from its non-linear processing, shows only a marginal improvement in handling the noise, as evidenced by its gradual but limited reduction in loss values.

The Deep Network shows a better approximation of the actual signal compared to the other models, as seen in the predictive performance plot in Fig.54. However, the corresponding loss curves for the Deep Network indicate a nuanced learning process. While the training loss drops quickly, the test loss displays considerable fluctuations before stabilizing. This suggests that while the network can learn the noise-affected patterns to an extent, its generalization to unseen data is not as effective, potentially due to some degree of overfitting to the noisy training data.

This analysis highlights that while deep learning models, particularly those with multiple layers, have a greater potential for capturing complex patterns, they also require careful tuning and possibly the integration of regularization strategies to mitigate the risks of overfitting under noisy conditions.

5 Tensor Decompositions for Big Data Applications

All answers for this section are presented in the form of Jupyter notebooks that are submitted.