Adaptive Signal Processing and Machine Intelligence

Department of Electrical and Electronic Engineering
& Department of Bioengineering

Imperial College London

Author:
Ciara Gibbs

CID:
01498482

April 2022

Contents

1 Classical and Modern Spectral Estimation

1.1 Properties of the Power Spectral Density (PSD)

As provided, there are two definitions for PSD:

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

and

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

The squared, absolute inner summation may be rewritten as the multiplication of a complex term with it's conjugate:

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

where * denotes the complex conjugate. Since the expectation operation is a linear operator, the summators may be factorised out of the expectation evaluation:

$$\lim_{N\to\infty} P(\omega) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} E\left\{x(k)e^{-j\omega k}x^*(m)e^{j\omega m}\right\}$$

(4)

$$\lim_{N\to\infty} P(\omega) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} E\left\{x(k)x^*(m)\right\}e^{-j\omega(k-m)} \tag{5}$$

The expectation operator can now be equated to autocovariance:

$$\lim_{N\to\infty} P(\omega) = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} r(k-m)e^{-j\omega(k-m)} \tag{6}$$

To simplify this expression for PSD, the double summation will be converted to a single summator, using a dummy variable, in which $\tau = k - m$. With this, the change in the lower and upper range of the summator operation, in terms of τ must also be considered.The range of the single summing operation is -(N-1) to (N-1). We must also consider the number of possible k-m combinations give a unique value of τ. It can be deduced that this is equal to $N - |\tau|$. This will serve as a multiplying factor to the new equation as seen below:

$$\lim_{N\to\infty} P(\omega) = \frac{1}{N}\sum_{\tau=-(N-1)}^{N-1} (N-|\tau|)r(\tau)e^{-j\omega(\tau)} \tag{7}$$

$$\lim_{N\to\infty} P(\omega) = \frac{1}{N}\sum_{\tau=-(N-1)}^{N-1} Nr(\tau)e^{-j\omega(\tau)} - \frac{1}{N}\sum_{\tau=-(N-1)}^{N-1} |\tau|r(\tau)e^{-j\omega(\tau)} \tag{8}$$

It is at this stage that we can now use the assumption the covariance sequence $r(\tau)$ decays rapidly. For

the term

$$\lim_{N\to\infty} \frac{1}{N} \sum_{\tau=-(N-1)}^{N-1} |\tau| r(\tau) e^{-j\omega(\tau)} \tag{9}$$

$r(\tau)$ decay means for most values of N, the expression is evaluated as ≈ 0. For the smallest of values of N, $\tau \approx 0$. Based on this overall, we can assume that:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{\tau=-(N-1)}^{N-1} |\tau| r(\tau) e^{-j\omega(\tau)} \approx 0 \tag{10}$$

Hence we approximate the power spectral density as

$$\lim_{N\to\infty} P(\omega) = \frac{1}{N} \sum_{\tau=-(N-1)}^{N-1} N r(\tau) e^{-j\omega(\tau)} = \sum_{\tau=-(N-1)}^{N-1} r(\tau) e^{-j\omega(\tau)} = \sum_{\tau=-\infty}^{\infty} r(\tau) e^{-j\omega(\tau)} \tag{11}$$

Therefore, it is shown that the definition 1 (equation 2) is equivalent to the definition 2 (equation 3) for PSD. Below it is shown that these two definitions are equivalent when the covariance sequence rapidly decays such as for sinusoidal signals, but are no longer equivalent if that assumption is violated i.e. the covariance sequence infact decays slowly, such as for impulses.

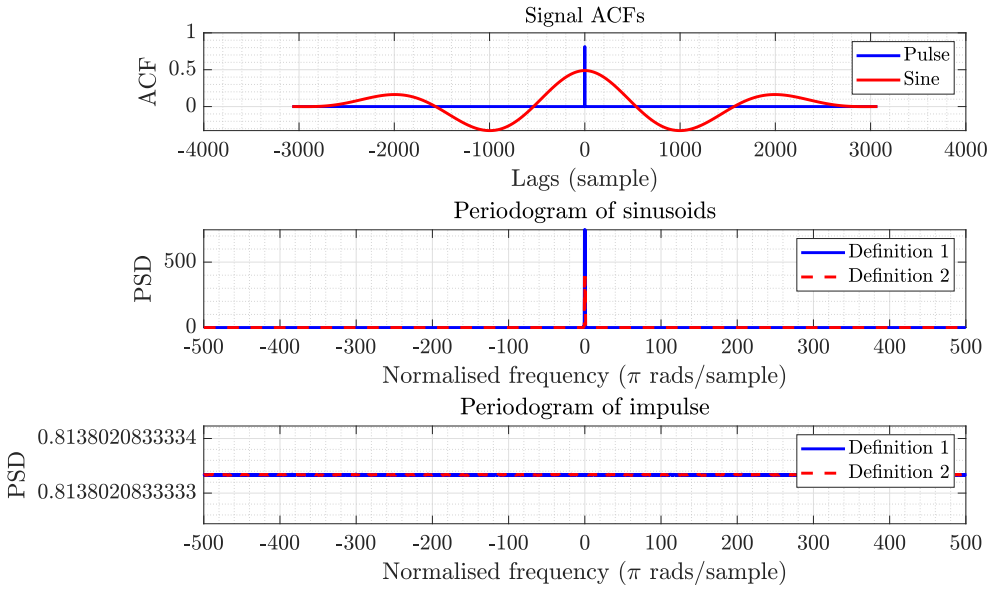


Figure 1: ACFs and PSD estimates of a sinusoid and impulse using definitions from equations 1 and 2 respectively.

1.2 Periodogram-based Methods Applied to Real-World Data

a) The periodogram for the sunspot data in MATLAB was determined through the application of both a rectangular window and Hamming window. Compared to the raw sunspot data, when the mean is subtracted the DC component of the data is removed, result in 0 power at 0Hz (figure axis does not display this in dB to avoid stretching). Detrend removes the best straight-fit line from the sunspot dataset. If the detrend and mean data had a significant difference in their power spectra, it would suggest that the mean of the signal is changing over time, such that detrend removes a trend with a non-zero gradient. As such it would imply non-stationary. However, in this instance, we see that the outcomes and mean subtraction and detrended are highly similar. In contrast, the application of a logarithm to the data,

followed by mean subtraction produces significantly different results to the prior two processing methods. Notably, to avoid logarithms of zero values, the numerical term in MATLAB `eps` $= 2.2204 \cdot 10^{-16}$ is added to the data, to make sure that zero terms are now non-zero. The logarithm compacts the data into a smaller scale; in doing so, the PSD is smoothed and useful peak information is seen more clearly.

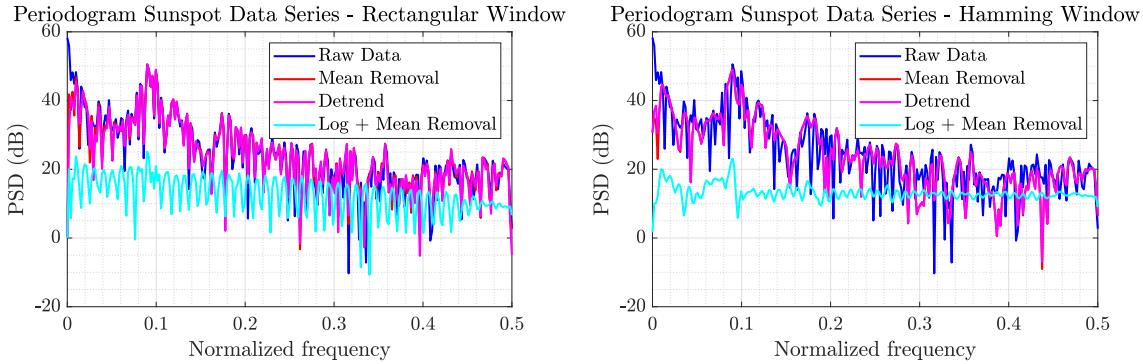


Figure 2: **Left**: Periodogram with a rectangular window applied to sunspot data. **Right**: Periodogram with a hamming window applied to sunspot data.

b) Standard and Bartlett methods are used to determine the periodogram of the provided EEG signal. As seen in figures 3 and 4 , the application of Bartlett's method via `pwelch`, the implementation of Welch's method in MATLAB, results in the revealing of several distinct peaks: a wide peak at 8-10Hz corresponding to subject fatigue, and narrow peaks at 50Hz, 13Hz, 26Hz and 39Hz, corresponding to DC noise, and the fundamental frequency of the SSEVP f_0^{SSEVP} and it's harmonics f_i^{SSEVP} respectively. The 3rd harmonic at 52Hz is difficult to identify for all windowing sizes due to the high-amplitude DC 50Hz mains component still present in the power spectra. When directly comparing the standard periodogram to 10s-length Bartlett's windowing method in figure 5, it is clear that windowing has reduced the variance in the PSD, but also adds bias to the outcome - which defines the bias-variance trade-off. This is further highlighted through direct comparison of the standard periodogram with the 1s-length Bartlett's windowing method. A smaller window length provides more segments over which the periodogram calculation is averaged, reducing the variance. This is evident by the increased output smoothness, but suffers the compromised of increased bias compared to the 10s windowing method.

Additionally, the 1s windowing eloquently demonstrates the different properties of rectangular and Hamming windowing methods. The Hamming window is a finite impulse response (FIR) filter, that gives a smoother and slightly lower amplitude output that the rectangular FIR filter, when comparing for the same length. The rectangular filter, particularly for 1s length has ripples about the large peak e.g. the 50Hz component due to it's intrinsic discontinuity in the temporal domain, causing energy leakage. In contrast, a Hamming window is continuous so does not result in the introduction of ripples in the PSD analysis. The marginally lower amplitude when applying the Hamming window is due to a reduction in signal energy.

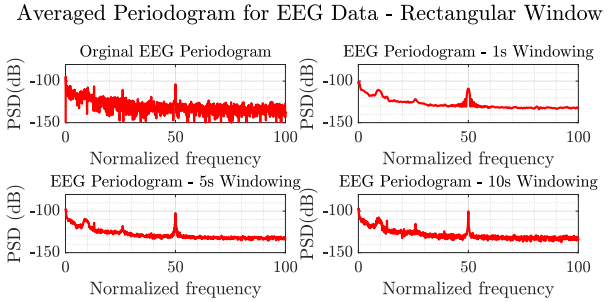


Figure 3: Periodogram applied to EEG data- Rectangular Window

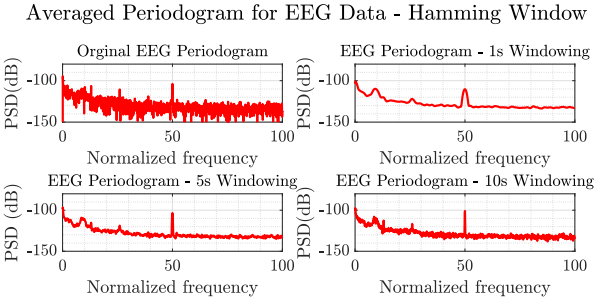


Figure 4: Periodogram applied to EEG data- Hamming Window

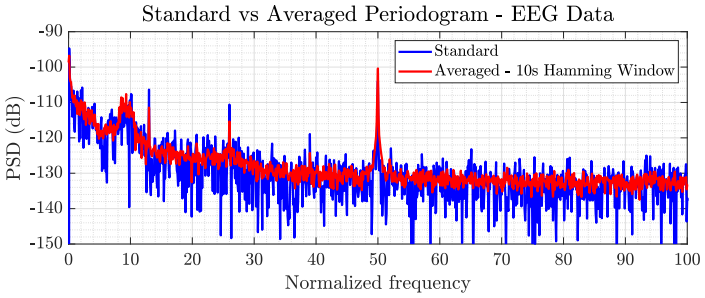


Figure 5: Standard vs Averaged Periodogram, EEG Data, Bartlett's Method with Hamming Windowing

1.3 Correlation Estimation

a) Both unbiased and biased ACF estimates of a signal were used to compute the correlogram, for white Gaussian noise (WGN), a noisy sinusoidal signal and filtered WGN. The biased and unbiased estimates are computed by:

$$Biased : \hat{r}(k) = \frac{1}{N} \sum_{n=0}^{N-|k|-1} x(n)x^*(n+k) \tag{12}$$

$$Unbiased : \hat{r}(k) = \frac{1}{N-|k|} \sum_{n=0}^{N-|k|-1} x(n)x^*(n+k) \tag{13}$$

As shwon in figure 6, both unbiased and biased ACF estimates when $|k| < 500$, the estimates are almost identical. However, beyond this, the unbiased and biased outputs increasingly differ; the biased ACF estimate decays to zero (reflecting the property that it is asymptotically unbiased) whilst the unbiased ACF estimate sees an increase in amplitude. When both these estimators are used for the correlogram (PSD), we see that the biased estimate provides a more accurate representation than the unbiased estimate. For our simulations we would expect: 1) **WGN**: flat spectrum 2) **Noisy sinusoid**: delta train with impulses corresponding to sinusoidal frequencies included 3) **Filtered WGN**: given that it is filtered with a low-pass filter, we expect that the spectrum has a lower amplitude overall compared to WGN, with (almost) complete removal of components corresponding to high normalised frequencies. The unbiased estimates do not reflect these features well. In particular, the unbiased estimates produces large negative amplitude components in all three simulations. The negative values in the unbiased ACF estimate arise from the nature of the computations performed. The unibiased estimate is equivalent to convolving the true correlogram/PSD with a rectangular window filter, whilst the biased estimate is based on convolution with a triangular Bartlett window. This is because in the time domain we can rewrite the biased and unbiased ACF estimators as:

where

$$w_{unbiased}(k) = \begin{cases} 1 & \text{for } |k| < N \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$$w_{biased}(k) = \begin{cases} \frac{N-|l|}{N} & \text{for } |k| < N \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Which would correspond to rectangular and Bartlett windows in the time domain, whereby multiplication in the time domain results in convolution in the Fourier domain. Given that a rectangular window is a periodic sinc function in the Fourier domain, this will introduce negative values that in turn will violate PSD being non-negative for the simulation cases we have tested. Finally, the denominator of the unbiased estimator '$N - |k|$' is responsible for the behaviour for large lag values (k). Combined with the upper bound of the summation, which means we use few values for estimates at large lags, it follows that the condition of positive definiteness may no longer hold, hence r(k) could be larger than r(0) which should

not occur. These reasons have lead overall to more common use of biased estimators for correlogram/PSD calculations.

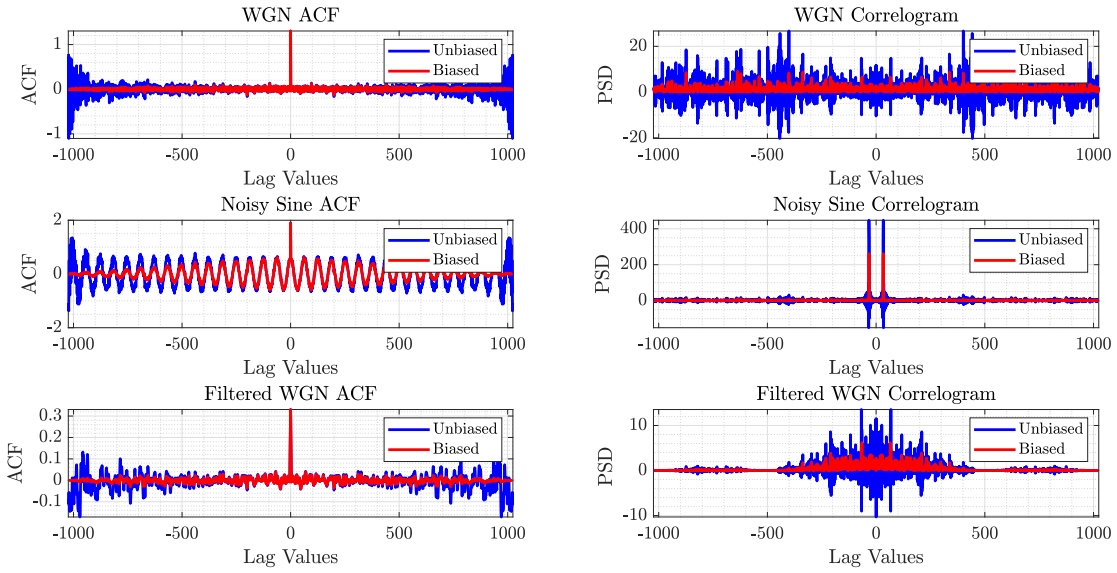


Figure 6: ACF and Correlograms (PSD) for WGN, noisy sinusoid and filtered WGN.

b) Using the previously coded biased ACF estimator, 100 realisations of a random process was simulated as described by a combination of three sinusoids, in addition to WGN:

$$x(n) = 1.5\sin(2\pi n1) + 0.8\sin(2\pi n0.3) + 0.6\sin(2\pi n1.5) + \eta(n) \tag{17}$$

Which should give rise to three peaks in the PSD spectrum at the frequencies 1Hz, 0.3Hz and 1.5Hz. As seen in figure 7, the average PSD across 100 realisations enables the identification of these peaks. We see some rippling around these peaks due to the nature of the biased ACF estimator. As previously mentioned, the Fourier domain representation of a biased ACF estimate corresponds to a triangular Bartlett window, for which it can be shown that it is equivalent to a squared, periodic sinc function. The associated spectral leakage creates these non-negative ripples. Furthermore, we see that the amplitude of the spectra further from the peaks converges to 1, which is the true amplitude of the PSD of WGN, hence demonstrating how the biased calculation of PSD is asymptotically unbiased.For the standard deviation plot of 7, we see that it takes a similar shape to the spectrum itself. In general, the PSD variance is difficult to compute since it depends on the fourth-order moment of the signal. However, as an approximate relation, the variance is proportional to the square of the PSD value at a given frequency point. Hence, significantly greater variance is both expected and seen at the peaks in the spectrum. As a result, the PSD estimator variance does not necessarily converge to 0 - hence it is an inconsistent estimator.

c) The same realisations as b) are now plotted on the decibel scale i.e. we take $10\log_{10}$ of the PSD values, shown in figure 8. The logarithm operators acts as a compressive function, such that it compresses the range of values. The gradient of a logarithm is $\frac{1}{x}$, therefore there is an inverse relationship between gradient of the dB representation of PSD and the PSD value at a given frequency point. Consequently, this attenuates the spectral peaks and relatively accentuates the noise peaks. In turn, the dB representation reduces the variance about the spectral peaks, whilst the variance of noise and low-PSD values experiences a relative increase in variance; overall this makes identifying the peaks of interest easier. Therefore, the dB representation of PSD is advantageous, particularly if we have a low-amplitude peak which would be more difficult to distinguish from noise in the non-dB transformed PSD plot.

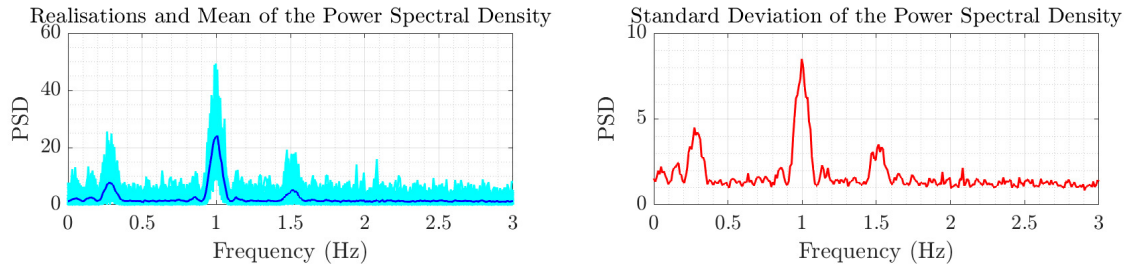


Figure 7: Plotting the PSD of a simulated noisy composite sinusoidal signal.

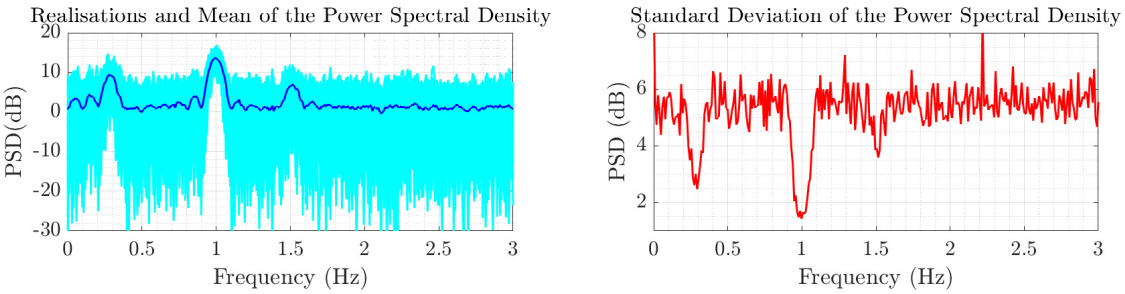


Figure 8: Plotting the PSD of a simulated noisy composite sinusoidal signal on a decibel scale.

d) The complex exponential signal provided in the coursework documentation was simulated for varying numbers of samples. By changing the sample number, the frequency resolution will in turn vary. It is shown in figure 9 that for sample length N=20 and N=35, the periodogram does not distinguish the two spectral peaks of the two frequencies. This is due to the resolution of the frequency axis being larger than the difference in the two frequency peaks. When $N \geq 45$ however, we see that the two peaks now become distinct, and not merged into one main lobe. We also see splitting at $N = 40$, but note that the peaks are wide and therefore not well centered about the true signal frequencies. The periodogram is derived from a biased estimate i.e. with a Bartlett window. Not only does the Bartlett window bias the periodogram, but it also introduces a smoothing effect, which limits the ability to resolve narrowband frequency components, that are present in our complex exponential signal. For a Barlett window, the resolution of the periodogram is approximately $\frac{0.89}{N}$, therefore, given the two frequencies of the signal:

$$\Delta f = \frac{0.89}{N} \rightarrow N = \frac{0.89}{\Delta f} = \frac{0.89}{0.32 - 0.3} = 44.5 \tag{18}$$

Hence, the minimum integer for sample length in theory to be able to resolve the two narrowband frequencies is 45, and this is corroborated by figure 9.

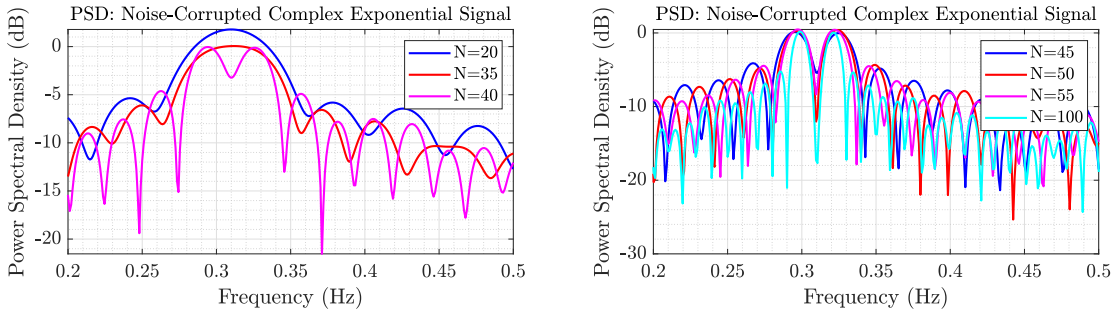


Figure 9: Simulating a complex exponential signal for varying sample lengths, with the standard periodogram method.

e) The **Mu**ltiple **Si**gnal **C**lassification Method (MUSIC) can be used find the line spectra of our simu-

lated complex exponential signal. It it classified as a subspace method, in which eigendecomposition is performed on the autocorrelation matrix of time series data.

```
[X,R] = corrmtx(x,14,'mod');
[S,F] = pmusic(R,2,[],1,'corr');
plot(F,S,'linewidth',2); set(gca,'xlim',[0.25 0.40]);
grid on; xlabel('Hz'); ylabel('Pseudospectrum')
```

First line: the function `corrmtx(x,14,'mod')` is used to derive a biased estimate of the autocorrelation matrix,R, of the input vector x (the complex exponential signal). The second input, 14, defines the prediction model order. This in turn defines the shape of the Toeplitz matrix X, that is used to determine the autocorrelation estimate R. $\boldsymbol{R} = \boldsymbol{X}^{\dagger}\boldsymbol{X}$ i.e. the autocorrelation estimate is equivalent to the matrix multiplication of the pseudo inverse of X, by X. Finally, the input 'mod' defines that the Toeplitz matrix X will be deduced from forward and backward errror estimates. This results in the rectangular form of X: 2(n-m) by (m+1), where n is the number of samples of the signal.
Second line: `pmusic(R,2,[],1,'corr')` is the in-built implementation of MUSIC and returns S, the psuedospectrum of the input vector x. The MUSIC algorithm assumes that the vector input x, is composed from complex exponentials, known as uncorrelated sources, as well as WGN. The output square autocorrelation matrix R from the first code line is provided as input to `pmusic`, which determines the eigenvalues of R and sorts them in descending order. The first i largest eigenvalues, where p is equal to to the number of complex exponentials that build the simulated signal vector x, represent the directions of greatest variability in the signal. Therefore, they are said to span the source subspace. Given that the model order is 14, the $14 - p$ eigenvalues left correspond to the eigenvectors that are said to span the noise subspace. The threshold between the assignment of eigenvalues to either subspace is set using the noise power σ_n^2: if the eigenvalues are greater than the noise power then they are associated with the source signal subspace. Moreover, this type of division into two subspaces is possible under assumptions about the autocorrelation matrix R. We assume that R can be split in the form:

$$\boldsymbol{R} = \boldsymbol{A}\boldsymbol{R_s}\boldsymbol{A^H} + \sigma_n^2\boldsymbol{I} \tag{19}$$

where A is and mxp matrix containing the p source subspace vectors $[e_1....e_p]$ where the complete set of signal vectors is $e_i = [1, e^{j2\omega}...., e^{j(m-1)\omega}]$. H is the Hermitian transpose of A and $\sigma_n^2\boldsymbol{I} = R_n$. These vector subspaces are orthogonal. The PSD derived by the MUSIC algorithm is given as:

$$\widehat{P_{music}(\omega)} = \frac{1}{\sum_{i=p+1}^{m} |e^H v_i|} \tag{20}$$

where $v_i (i = 1, 2, ...p)$ are the noise eigenvectors. Since the inner product between the source signal and noise eigenvectors will be 0, the PSD MUSIC-based estimate will result in p spectral peaks.The other inputs to `pmusic` are : 2 = p which is the dimensionality of the source signal subspace, [] sets the fft default length of 256, 1 denotes the sampling frequency, and 'corr' denotes that our input matrix is an autocorrelation matrix estimate. **Third line:** Plotting the frequency spectrum output where F is the frequency axis and S is the PSD. The x axis is bounded in the range of 0.25-0.40Hz.

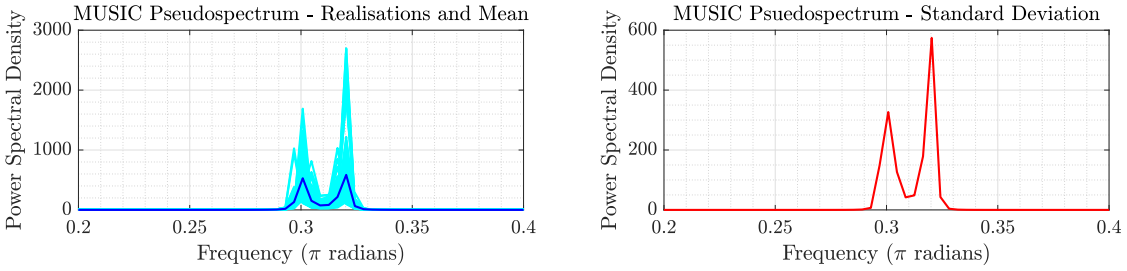


Figure 10: Pseudospectrum of simulated complex exponential using the MUSIC method.

The PSD resulting from the MUSIC algorithm is depicted in figure 10, for sample length N = 30. As

shown, MUSIC is capable for resolving to two spectral peaks for this small sample length, whilst the standard periodogram was incapable. Hence, MUSIC is advantageous over the standard periodogram in terms of resolution, reducing bias. However, there are important disadvantages. Primarily, the method relies on the user knowing the model order p i.e. the dimension of the source signal subspace, which is unlikely to be known apriori; selecting the wrong p value may lead to rapid deterioration in the PSD estimate. Furthermore, the variance of the estimate is significantly higher, despite the higher resolution. This in turn affects the reliability of the peak magnitudes; so although you can localise the peaks, their relative magnitudes may not be accurate. Overall, MUSIC is useful for PSD analysis is you have few samples, although will likely require additional work for estimating the model dimensions, such as using the Akaike Information Criterion (AIC) or Minimum Description Length (MDL).

1.4 Spectrum of Autoregressive Processes

a) Conventionally, the parameters of an autoregressive (AR) process may be determined using the vector-matrix form of the Yule-Walker equations, whereby $\boldsymbol{r_{xx}} = \boldsymbol{R_{xx}}\boldsymbol{a}$ and $\boldsymbol{R_{xx}}$ is the autocorrelation function (ACF) matrix of the signal of interest, $\boldsymbol{r_{xx}}$ autocorrelations and $\boldsymbol{a}$ the AR parameters. To able to obtain the AR parameters, $\boldsymbol{R_{xx}}$ must be positive definite (Toeplitz), such that matrix inversion is guaranteed. In figures , we compare biased and unbiased estimates of the ACF; for small lags the estimates are similar, whilst as the lag increases the biased ACF estimate tends to zero and the unbiased ACF estimate remains non-zero. Hence, if we have an unbiased ACF estimate, R_{xx} will no longer be positive definite, and hence we cannot invert the matrix to derive (stable) solutions of AR parameters $\boldsymbol{a}$.

b and c) The ARMA process $x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n)$ was implemented in MATLAB where w represents Gaussian noise with mean 0 and variance 1, using the function `filter(1,a, randn(n,1))` where a is an array of the ARMA process weights: `a = [1, -2.76, 3.81, -2.65, 0.92]`. With this, the power spectral density was estimated with `freqz` , which returns a n-point frequency response, that can be transformed to power. Firstly, sample legngth n was set to 1000 and was later adjusted to n = 10000, seen in figure 11 (note the first 500 samples are discarded to remove transient filter effects). As expected AR(2) is unable to effectively model the true PSD, as the lack of parameterisation results in only a single peak in the power spectrum. Interestingly, for n = 1000, AR(4) is also unable to model the two peaks; this suggests a greater resolution in the power spectra calculation is also important, not only the order of the model used. The higher order model AR(11) as shown in the figure was sufficiently parameterised to model the two peaks in the spectra. giving the lowest MSE. Notably, the tails of these models are not able to fit as well to the true power spectrum, as they are higher parameterised to fit the complexity of the two peaks, and therefore overfit to this region. When the number of signal sample points are increased to 10000, while AR(2) again fails to model the two-peak spectrum, the AR(4) model which is the true order is now capable of representing both peaks in the spectra, and with lowest MSE. Increasing model order gives no further improvement in estimation performance.

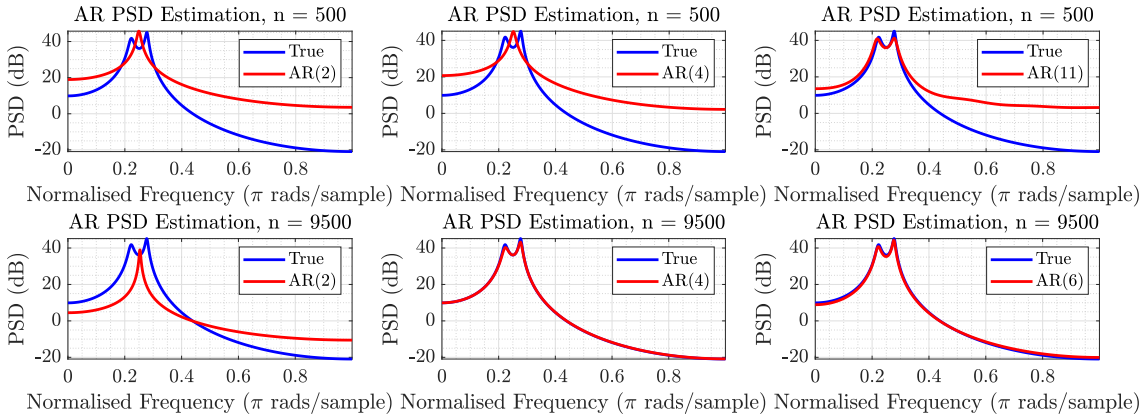


Figure 11: Modelling of a AR(4) process using 1000 (top) and 10000 (bottom) sample signal lengths.

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

a) The standard and averaged periodograms for the RR intervals, the time passed between two consecutive R-waves of the QRS complex in ECG data, was plotted calculated and plotted in figure 12. The averaged periodograms were determined in MATLAB using the function `pwelch`, the implementation for Welch's overlapped segment averaging estimator. Inputted to this function where rectangular window forms of temporal lengths $10s, 50s, 100s, 150s$, colour coded in figure 12.

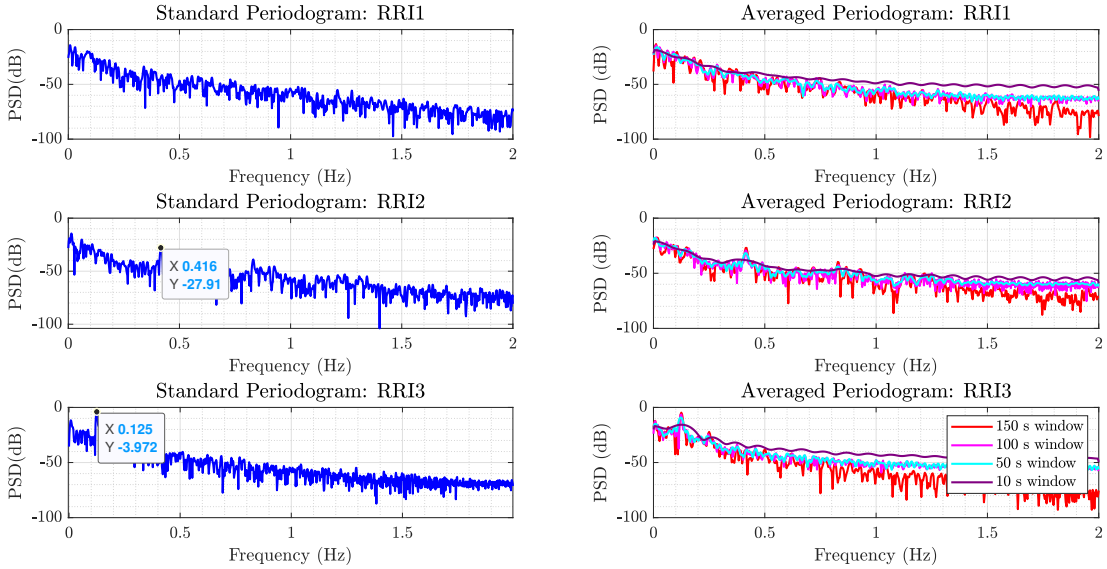


Figure 12: Standard (left) and Averaged (right) periodograms for RRI data.

b) **RRI differences**: The peaks in the spectra are located in difference places for the difference RRI instances. For RRI1 (and referring to the standard periodogram), no distinguishable peak is seen which can correspond to unconstrained breathing. For RRI2, a peak occurs at 0.416 Hz (3.d.p) equivalent to 50bpm , which corresponds to fast breathing. Finally for RRI3, a peak occurs at 0.125 Hz (3.d.p) equivalent to 14.76bpm which corresponds to slow breathing. These are the primary frequency components of the RRI signals, but harmonics are also reasonably noticeable in RRI2 and RRI2 periodograms (approximately 0.83Hz and 0.25Hz).

Window length differences: The different windowing lengths have varying effects on the output. Since the windows were set through the function `pwelch` to be non-overlapping, it is equivalent to using Bartlett's method. With this method, smaller window lengths decrease variance but increase bias. This is because for smaller window lengths, there are more segments available to average over. This is evident when comparing the smallest and largest window lengths tested - 10s and 150s - in which for the 10s window length the PSD is significantly smoother, but the tail of the spectra is significantly higher that the standard periodogram and that of the results from larger window lengths.

c) AR spectrum estimates for the RRI signals for the three trials are plotted in figure 13. For RRI3, AR(25) only appears capable of representing the fundamental frequency and subsequent harmonics. For RRI2, the AR(11) model is capable of representing three peaks - the fundamental frequency and two harmonics. Finally, for RRI1, AR(8) was used as a balance between model accuracy and overfitting. All of these corroborate the periodograms, but can aid in visualistion of peaks since they are representing where the noise power is negligble in comparison to the power of the signal. It is important to make sure that the AR model selected is not over-parameterised and hence overfitting to noise, which may be the case for AR(25) when used in RRI3 modelling. Despite this, they can still be advantageous compared to periodogram averaging, since they do not suffer from bias/resolution-variance trade-offs.

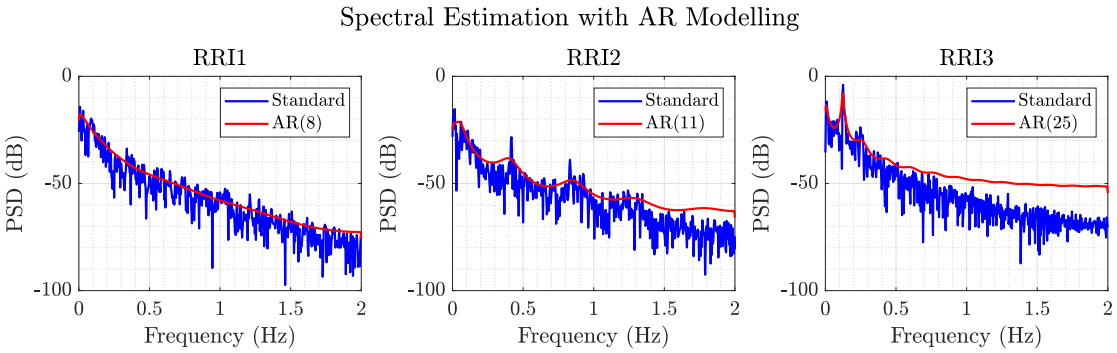


Figure 13: Standard (left) and Averaged (right) periodograms for RRI data.

1.6 Robust Regression

a) The singular values for $\boldsymbol{X}$ and $\boldsymbol{X_{noise}}$ are plotted in figure 14 respectively. From this, it is clear that the rank of $\boldsymbol{X}$ is since only the first 3 singular values (eigenvalues) are non-zero. In contrast, the singular values corresponding to subspace dimensions greater than 3 are non-zero, although it can still be seen that the first 3 singular values are significantly larger. The squared error between $\boldsymbol{X}$ and $\boldsymbol{X_{noise}}$ are then plotted in figure **??**. This shows that for the first three subspace dimensions the squared error remains low, but beyond this there is a significant increase in error. Therefore, beyond the third dimension, it may become difficult to distinguish the true rank of the signal $\boldsymbol{X}$.

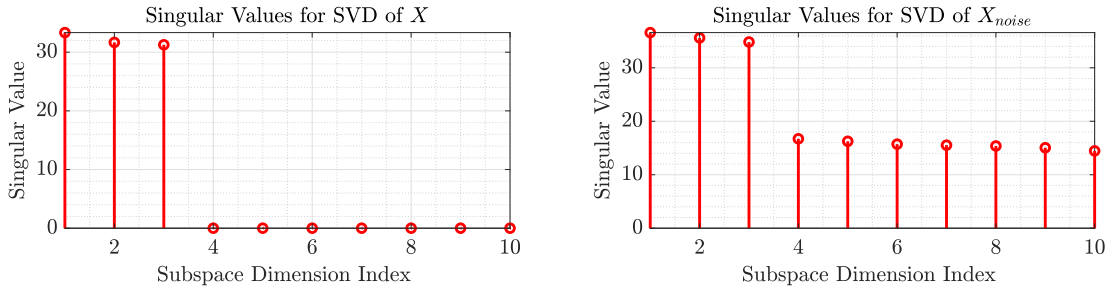


Figure 14: Singular values of noiseless matrix $\boldsymbol{X}$ (left) and noise corrupted matrix $\boldsymbol{X_{noise}}$ (right).

b) Having deduced that the rank is 3, principal component analysis (PCA) is performed on $\boldsymbol{X_{noise}}$ using only the first 3 components (the most significant), to create a low-rank approximation, $\boldsymbol{\widetilde{X_{noise}}}$. The squared error between $\boldsymbol{X}$ and the de-noised matrix $\boldsymbol{\widetilde{X_{noise}}}$, as well as $\boldsymbol{X}$ and noise-corrupted $\boldsymbol{\widetilde{X_{noise}}}$ was calculated and plotted in figure 15. Unsurprisingly the squared error between $\boldsymbol{X}$ and $\boldsymbol{\widetilde{X_{noise}}}$ is significantly larger due to the additional non-zero singular values in subspace dimensions greater than 3.

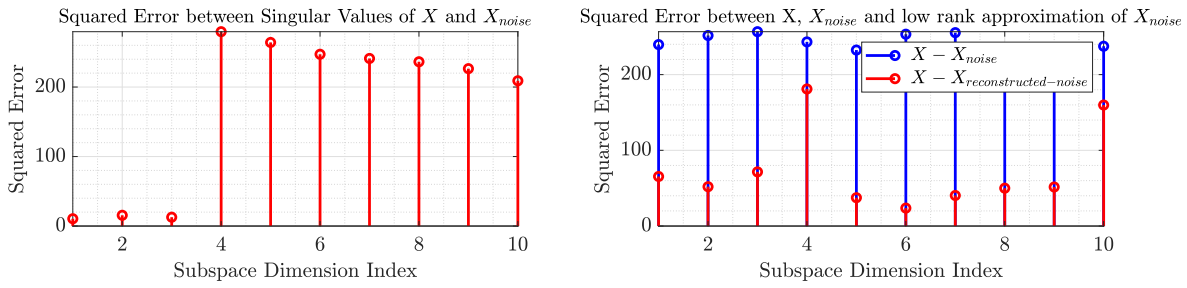


Figure 15: Squared error between noiseless and noise matrices (left) and noiseless and de-noised matrices (right) .

c) The output data is derived using the equation $\boldsymbol{Y} = \boldsymbol{XB} + \boldsymbol{N_Y}$. The matrix $\mathbf{B}$ is a parameter matrix which relates the noise matrix to the output $\mathbf{Y}$ and can be obtained with two methods 1) Ordinary Least Squares and 2) Principal Component Regression. Complete/valid OLS-based determination of $\mathbf{B}$ is achieved using:

$$\widehat{\boldsymbol{B_{OLS}}} = (\boldsymbol{X_{noise}}\boldsymbol{X_{noise}^T})^{-1}\boldsymbol{X_{noise}^T}\boldsymbol{Y} \tag{21}$$

where $\boldsymbol{X_{noise}}$ is used in-place of $\boldsymbol{X}$ as it is full-rank and hence the inverse remains a valid operation. For PCR:

$$\widehat{\boldsymbol{B_{PCR}}} = (\boldsymbol{V_{1:r}}(\boldsymbol{\Sigma_{1:r}}))^{-1}\boldsymbol{U_{1:r}^T}\boldsymbol{Y} \tag{22}$$

where $\mathbf{V}$, Σ and $\mathbf{U}$ are components of a singular value decomposition (SVD) as part of PCA, considering only the r largest principcal components (singular values). The output is then calculated with each of these matrix estimators with equations $\mathbf{Y_{OLS}} = \widehat{\mathbf{B_{OLS}}}\mathbf{X_{noise}}$ and $\mathbf{Y_{PCR}} = \widehat{\mathbf{B_{PCR}}}\widetilde{\mathbf{X_{noise}}}$ respectively. From this, the estimate error between these outputs and the in-sample/directly used and out-sample/test data was calculated and plotted in figure 16. Narrowly, the squared error for each subspace dimension is larger for PCR for the in-sample data, and then marginally smaller for the out-sample/test data. This, again narrowly, suggests that the OLS operation is susceptible to overfitting to the noise in the input data.

d) To robustly compare the effectiveness of PCR against OLS, the matlab function `regval` was used to generate a realisations of test data, $\mathbf{Y}$, along with their estimates $\widehat{\boldsymbol{Y}}$. This permits the calculation of mean-squared error (MSE) for OLS and PCR-based methods over an ensemble of data. The MSE is plotted in figure 16. This corroborates and depicts more clearly that PCR performs better on the test/out-sample data. However, this marginal improvement in performance should in practice be considered alongside computational power and processing time, as PCA may become a lengthy calculation for higher-rank matrices.

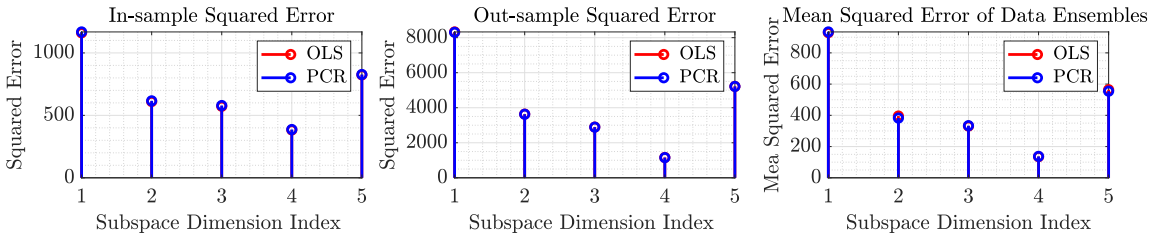


Figure 16: Error analysis over ensembles of test and estimator data generated via OLS and PCR methods.

2 Adaptive Signal Processing

2.1 Least Mean Square (LMS) Algorithm

a) In this question we are provided with an expression for signal $x(n)$, a second-order auto-regressive process satisfying the difference equation:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n) \qquad \eta(n) \sim N(0, \sigma_\eta^2) \tag{23}$$

where the original parameters are given as $a_1 = 0.1$, $a_2 = 0.8$ and $\sigma_\eta^2 = 0.25$. We will implement adaptive linear prediction using the LMS algorithm. Firstly, the inputs to the LMS filter are $x(n-1)$ and $x(n-2)$, creating an input vector:

$$\boldsymbol{x}(n) \equiv [x(n-1), x(n-2)]^T \tag{24}$$

The correlation matrix, based on our input vector is derived through the following steps:

$$\boldsymbol{R_{xx}} = \mathbb{E}[\boldsymbol{x(n)}\boldsymbol{x(n)^T}] = \begin{bmatrix} x(n-1)x(n-1) & x(n-1)x(n-2) \\ x(n-2)x(n-1) & x(n-2)x(n-2) \end{bmatrix} \tag{25}$$

which written in the form of the autocorrelation matrix is equivalent to:

$$\boldsymbol{R_{xx}} = \begin{bmatrix} r_{xx}(0) & r_{xx}(1) \\ r_{xx}(1) & r_{xx}(0) \end{bmatrix} \tag{26}$$

In order to deduce the values of these autocorrelation expressions, we use the general form the autocorrelation function for a AR(2) process. This requires the multiplication of the signal $\boldsymbol{x(n)}$

$$r_{xx}(k) = \mathbb{E}[x(n)x(n-k)] = \mathbb{E}[a_1x(n-1)x(n-k) + a_2x(n-2)x(n-k) + \eta(n)x(n-k)] \tag{27}$$

and by the linearity of the expectation operator:

$$r_{xx}(k) = a_1\mathbb{E}[x(n-1)x(n-k)] + a_2\mathbb{E}[x(n-2)x(n-k)] + \mathbb{E}[\eta(n)x(n-k)] \tag{28}$$

With this, we can derive the expressions for $r_{xx}(0)$ and $r_{xx}(1)$:

$$r_{xx}(0) = a_1r_{xx}(1) + a_2r_{xx}(2) + \sigma_\eta^2 \tag{29}$$

$$r_{xx}(1) = a_1r_{xx}(0) + a_2r_{xx}(-1) = a_1r_{xx}(0) + a_2r_{xx}(1) \tag{30}$$

where σ_η^2 is the variance of the signal noise, and is only included in $r_{xx}(0)$ as $+\mathbb{E}[\eta(n)x(n-k)] = 0$ when $k > 0$. Since the expression for $r_{xx}(0)$ also includes r_{xx}, we must also derive it's expression:

$$r_{xx}(2) = a_1r_{xx}(1) + a_2r_{xx}(0) \tag{31}$$

thus providing three equations for which we can solve for the three unknowns $r_{xx}(0)$, $r_{xx}(1)$ and $r_{xx}(2)$, whilst using the values $a_1 = 0.1$, $a_2 = 0.8$ and $\sigma_\eta^2 = 0.25$. Solving this gives 1) $r_{xx}(0) = \frac{25}{27}$ and $r_{xx}(1) = \frac{25}{54}$. The autocorrelation matrix is therefore equal to:

$$\boldsymbol{R_{xx}} = \begin{bmatrix} \frac{25}{27} & \frac{25}{54} \\ \frac{25}{54} & \frac{25}{27} \end{bmatrix} \tag{32}$$

The convergence condition for the LMS algorithm is parameterised by the step size μ. In order to achieve convergence to the optimal Wiener solution, μ must satisfy the condition:

$$0 < \mu < \frac{2}{\lambda_{max}} \tag{33}$$

for which λ_{max} is the largest eigenvalue of the autocorrelation matrix $\boldsymbol{R_{xx}}$. Through eigen-analysis, it can be deduced that $\lambda_{max} = 1.389(3.d.p)$ and hence $\mathbf{0 < \mu < 1.440}$ (3.d.p).

b) The LMS adaptive predictor for 1000 samples of signal $x(n)$ was implemented in `MATLAB`, using two step sizes, 1) $\mu = 0.01$ and 2) $\mu = 0.05$. The squared prediction error between the signal $x(n)$ and the corresponding LMS model $\hat{x}(n)$ was computed as $e^2(n) = (x(n) - \hat{x}(n))^2$. The squared prediction error was then evaluated for one iteration, and one hundred, where the latter reduces the variance and therefore allows us to better infer the convergence properties of the LMS algorithm for the two step sizes tested. These learning curves are depicted in figure 17.

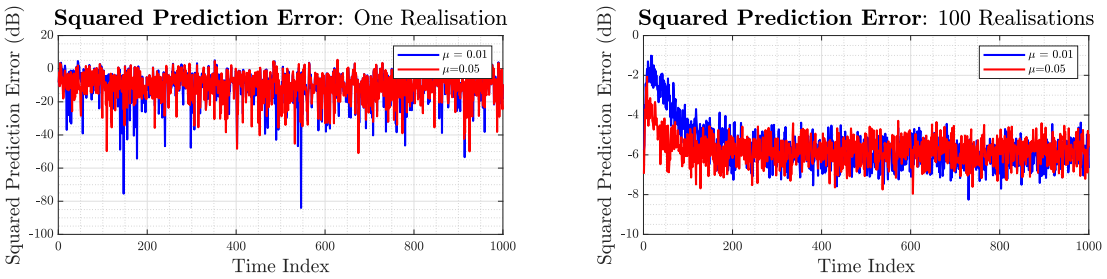


Figure 17: Squared prediction error in decibels from an LMS adaptive predictor for 100 samples of signal $x(n)$. Left: One realisation, Right: 100 realisations.

As shown in the figure above, only after averaging across one hundred iterations can we effectively analyse how the prediction error converges for different step sizes. Hence, in the right figure, it is clear that the convergence speed is greater for $\mu = 0.05$, converging to a steady-state value approximately twice as fast as $\mu = 0.01$ ($\approx$ 100 vs 200 samples). However, for $\mu = 0.05$, the steady-state squared prediction error value is marginally higher compared to when using $\mu = 0.01$.

c) Excess error is introduced to the minimum achievable error signal as a result of LMS weights varying about the optimal Wiener solution (the Wiener-Hopf solution). The excess mean square error (EMSE) metric defines the ratio between the MSE originated from the adaptive filters, plus the minimum achievable signal associated with a Wiener filter. Collecting this into an equation gives:

$$MSE = \lim_{n\to\infty} \mathbb{E}\{e^2(n)\} = \sigma_\eta^2 + EMSE \tag{34}$$

Furthermore, the **theoretical** misadjustment $\mathcal{M}_{LMS}$ is defined as the ratio of ESME to minimum MSE, but for small step sizes/learning rates, it is approximated as:

$$\mathcal{M}_{LMS} \approx \frac{\mu}{2} Tr\{\mathbf{R}_{xx}\} = \frac{\mu}{2} = \frac{\mu}{2} 1.\dot{8}5\dot{1} \tag{35}$$

where $Tr\{.\}$ is the matrix trace operator, $\mathbf{R}$ the autocorrelation matrix and μ the learning rate. Calculating the trace from equation 36 gives a value of 1.8519 Despite the fact that the learning curves simulated for $\mu = 0.05$ and $\mu = 0.01$ have both converged by observation around the time index ~ 200, a buffer window is still used for accuracy in determining a time average for the MSE. Hence, an average is computed across time using time indices 600-1000. With this, we can determine the **empirical** misadjustment via the following equation:

$$\mathcal{M}_{LMS} = \frac{EMSE}{\sigma_\eta^2} = \frac{MSE - \sigma_\eta^2}{\sigma_\eta^2} = \frac{MSE}{\sigma_\eta^2} - 1 \tag{36}$$

The empirical and theoretical values are shown in table 1.

μ	$\mathcal{M}_{empirical}$	$\mathcal{M}_{theoretical}$
0.01	0.0099	0.00926
0.05	0.0567	0.04630

Table 1: LMS empirical and theoretical misadjustments, computed with signal $x(n)$.

The mismatch between the empirical and theoretical derivations of $\mathcal{M}_{LMS}$ could be due to several factors 1) The approximation use for small step-sizes when deriving the theoretical estimate 2) Use of a finite-sized signal across time, but also from an ensemble perspective and 3) Estimation of the time index for convergence i.e. taking time indices 600-1000 for time-averaging. Additionally, $\mathcal{M}_{LMS}$ is larger in both empirical and theoretical derivations for $\mu = 0.05$, suggesting therefore that it introduces excess error to the minimum achievable error signal (as set by the Wiener filter).

d) The steady-state values of the filter weights were estimated by averaging 100 realisations of LMS implementation, and the averages plotted in figure 18 when using step sizes $\mu = 0.01$ and $\mu = 0.05$. For step size of 0.01,$\hat{a}(1)$ converges to the true value (with a small negative offset of 0.004), $a(1)$, whilst at step size 0.05, there is a negative offset of 0.023. However, $\hat{a}(2)$ is lower than $a(2)$ both step sizes, where $\mu = 0.01 \rightarrow \hat{a}_{steadystate}(2) = 0.780$ giving a 0.02 negative offset, and $\mu = 0.05 \rightarrow \hat{a}_{steadystate}(2) = 0.719$, give a 0.81 negative offset. Therefore overall, using a larger step size decreases estimation performance. This is because although larger step sizes increase the speed of convergence, they increase the variance in the estimation process, hence a trade off arises between convergence speed and steady state error.

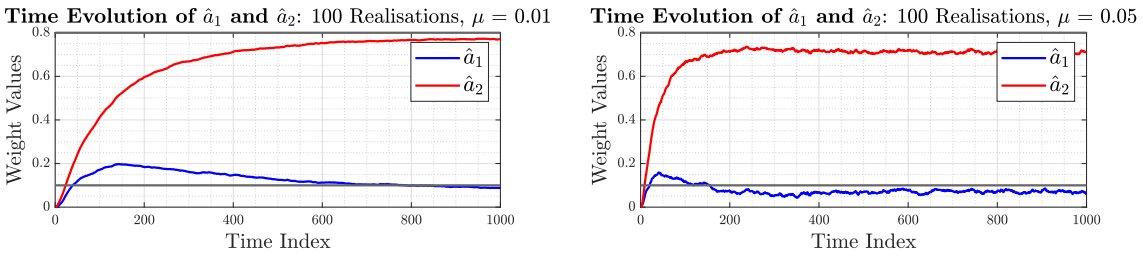


Figure 18: Evolution of filter weights in LMS for learning rates $\mu = 0.01$ and $\mu = 0.05$.

e) If the provided input signal $x(n)$ has zero-valued eigenvalues in the corresponding autocorrelation matrix $\boldsymbol{R_{xx}}$, then the LMS adaptive filter may suffer from instability due to a lack of convergence in the filter weights. To promote stabilisation, leakage coefficient γ is introduced. Given the cost function criteria J_2, such that:

$$J_2(n) = \frac{1}{2}\Big(\boldsymbol{e}^2(n) + \gamma\|\boldsymbol{w}(n)\|_2^2\Big) \tag{37}$$

The error term can be rewritten in terms of the signal $\mathbf{x}(n)$ and the desired value $y(n)$, shown in appropriate matrix notation:

$$J_2(n) = \frac{1}{2}\Big((\boldsymbol{y}(n) - w(n)^T\boldsymbol{x}(n))^T(\boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)\big) + \gamma\boldsymbol{w}(n)^T\boldsymbol{w}(n)\Big) \tag{38}$$

Following from this, the gradient of the cost function $\nabla_w J_2$, is computed, differentiating with respect to weights $\boldsymbol{w}(n)$:

$$\nabla_w J_2(n) = \frac{1}{2}\Big(- \boldsymbol{x}(n))^T(\boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)) - \boldsymbol{x}(n))^T(\boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)) + 2\gamma\boldsymbol{w}(n)\Big) \tag{39}$$

$$\nabla_w J_2(n) = \frac{1}{2}\Big(- 2\boldsymbol{x}(n))^T(\boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)) + 2\gamma\boldsymbol{w}(n)\Big) = \Big(\boldsymbol{x}(n))^T(\boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)) + \gamma\boldsymbol{w}(n)\Big) \tag{40}$$

$$\nabla_w J_2(n) = \Big((\boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n))\boldsymbol{x}(n) + \gamma\boldsymbol{w}(n)\Big) \tag{41}$$

The gradient descent update to derive the minimum of the cost function $J_2(n)$ is defined as:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu\nabla_w J_2(n) \tag{42}$$

Substituting back into our expression that $\boldsymbol{e}(n) = \boldsymbol{y}(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)$ and in turn substituting the gradient of the cost function:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu\big(- \boldsymbol{e}(n)\mathbf{x}(n) + \gamma\boldsymbol{w}(n)\big) = (1 - \mu\gamma)\boldsymbol{w}(n) + \mu\boldsymbol{e}(n)\boldsymbol{x}(n) \tag{43}$$

Hence, the LMS leaky equation, using the LMS coefficient update and the provided cost function, is derived, matching the expression provided.

f) The leaky LMS algorithm was implemented in `MATLAB` for different values of the learning rate μ and the leakage factor γ. The simulations are shown in figures —.

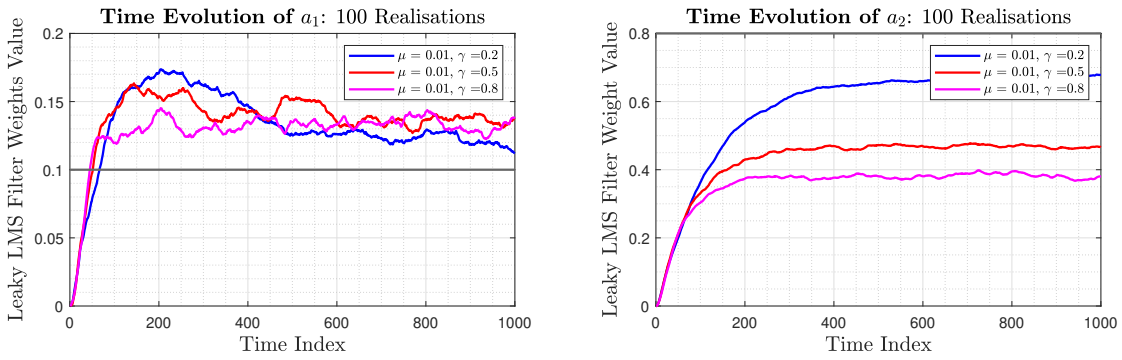


Figure 19: Evolution of filter weights in LMS for learning rates $\mu = 0.01$ and $\gamma = [0.2, 0.5, 0.8]$.

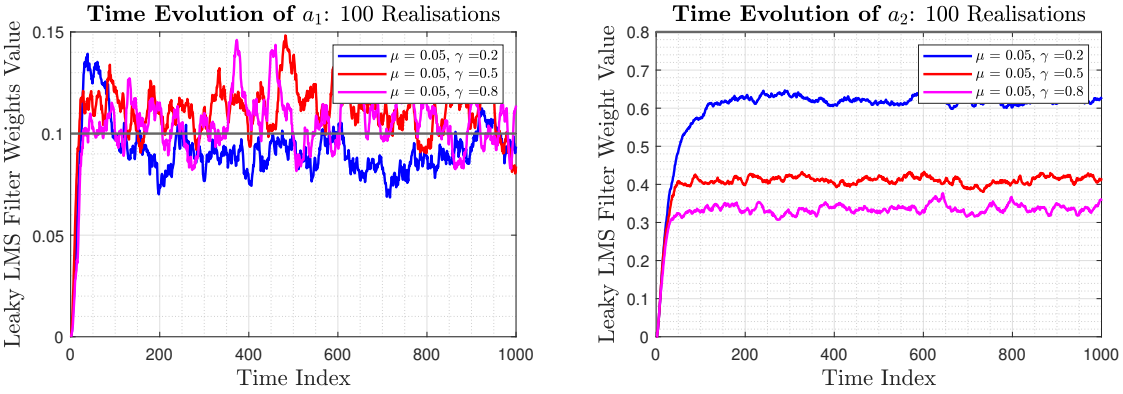


Figure 20: Evolution of filter weights in LMS for learning rates $\mu = 0.05$ and $\gamma = [0.2, 0.5, 0.8]$.

As seen in figures 19 and 20,in the 1000 time steps, the filter weights fail to converge to a steady-state value that equals the coefficients a_1 and a_2. Furthermore, the larger the leakage coefficient γ, the greater the bias in the coefficient estimates. The relationship between the leakage coefficient γ and the bias in a_1/a_2 estimates can be explained by referring to the Wiener-Hopf solution:

$$\boldsymbol{w} = \boldsymbol{R}_{\boldsymbol{xx}}^{-1}\boldsymbol{p} \tag{44}$$

where $\boldsymbol{R_{xx}}$ is the autocorrelation matrix and p the cross-correlation vector computed between the input signal $\boldsymbol{x(n)}$ and desired value $\boldsymbol{d(n)}$. As previously noted, in the instance where the autocorrelation matrix has zero-valued eigenvalues, the LMS filter may lack instability. This is because $\boldsymbol{R_{xx}}$ is no longer a positive-definite matrix, but positive semi-definite, hence may not be invertible. Consequently to mitigate this, the leakage coefficient is introduced, which alters equation 48 to:

$$\boldsymbol{w_{leaky}} = (\boldsymbol{R_{xx}} + \boldsymbol{\gamma I})^{-1}\boldsymbol{p} \tag{45}$$

Hence as γ tends to 0, equation 49 converges to the Wiener-Hopf solution, meaning larger values of γ will result in increasing bias relative to this Wiener-Hopf solution. Not only this, bias may incur from the perspective of the weight update function, since γ reduces the weighting of $\boldsymbol{w}(n)$ in the update rule for computing $\boldsymbol{w}(n+1)$ (equation 47). Additional noteworthy features are 1) The variance in the estimates of a_1 and a_2 increases with the step-size/learning rate μ and 2) The bias in the estimate of a_2 is worse than that of a_1 for all tested parameterisations of the leaky LMS model. Feature 1)is likely due to larger 'jumps' in learning at each step while 2) is likely due to the respective magnitudes of the coefficients, where $a_2 > a_1$.

2.2 Adaptive Step Sizes

a) Given the real-valued MA(1) system $x(n) = 0.9\eta(n-1) + \eta(n), \eta \sim \mathcal{N}(0, 0.5)$ if the LMS algorithm is applied to approximate the Wiener solution, a trade-off between steady state variance and convergence speed will always arise given a fixed value for the step-size μ. Gradient Adaptive Step-Size (GASS) algorithms are designed to adapt the step size μ via gradient-based minimisation of a cost function, $\Delta_\mu J$, such that benefits large $\mu's$ (convergence) and small (steady-state error) can be exploited. Three different GASS algorithms 1) Benveniste, 2) Ang and Farhang and 3) Matthews and Xie, are to be evaluated based on weight error curves for which $\tilde{w}(n) = w_o - w(n)$ (where $w_o = 0.9$) is plotted.

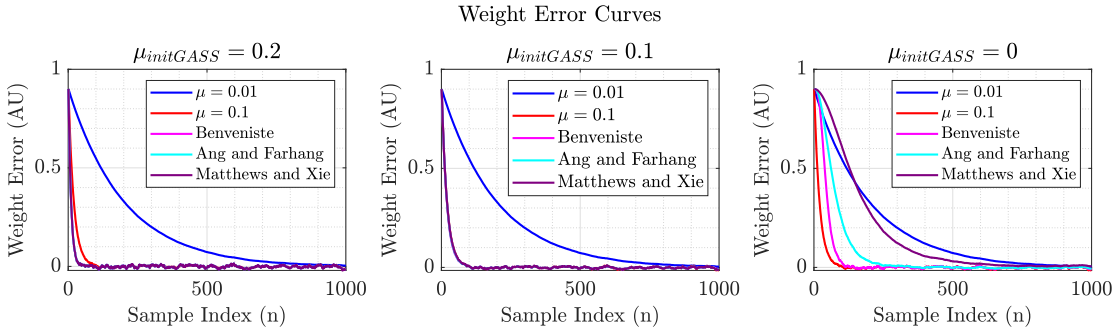


Figure 21: **Left**: Weight Error curve with GASS step-size intialised as 0.2. **Middle**: GASS step-size intialised as 0.1. **Right**: GASS step-size intialised as 0.

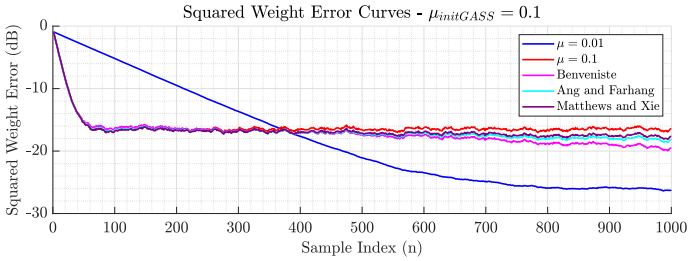


Figure 22: Squared Weight Error Curve ($10 \log 10(mean(weight - error^2))$) when GASS is initialised with $\mu = 0.1$.

As displayed in figure 21 when the GASS algorithms are initialised with a step-size of $\mu = 0.1$, all 3 algorithms perform better than a constant value of step-size across time equal to 0.01 but what appears to be the same when the constant is set to 0.1. However, upon closer inspection in figure 22, we see clearly that the GASS algorithms converge to a steady state (error) much faster than when using a constant step size of 0.1. However, the steady state for GASS algorithms is greater. Notably, Benveniste's algorithm has the lowest steady error from the GASS algorithms of -19.6dB (compared to -17.9dB for Ang and Farhang and -17.2dB for Matthews and Xie) likely since it has the greatest computational complexity. The superior performance of Benveniste's algorithm is also evident when the GASS algorithms are initialised with $\mu_{init} = 0$, for which the convergence speed of Benveniste's is greatest from the GASS algorithms. Benveniste's algorithm has a computational complexity of $O(M^2)$, where M is the model order, directly attributed to calculating the sensitivity term ψk. $\psi(k)$ which filters noisy instantaneous gradients, where for the update $\psi(n) = [\mathbf{I} - \mu(n-1)\mathbf{x}(n-1)\mathbf{x}^{\mathbf{T}}(n-1)]\psi(n-1) + e(n-1)\mathbf{x}(\mathbf{n}-\mathbf{1})$, the time-dependent term in square bracket directly influence the filtering properties. Ang and Farhang's algorithms reduces complexity by substituting the square-bracketed term to the term α, while Matthews and Xie fixes α to 0, removing the first term completely. This sequential reduction in complexity explains the sequence of convergence speed seen for $\mu_{init} = 0$, and the steady errors of figure 22, where from GASS algorithms the lowest steady-state error is Benveniste's, second Ang and Farhang, and last Matthews and Xie.

b) To verify that the update equation $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e_p(n)x(n)$ based on the a posteriori error $e_p(n) = d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n+1)$, is equivalent to the Normalised Least Mean Squares algorithm (NLMS),

we begin by multiplying both sides of the update equation by $\boldsymbol{x}(n)^T$:

$$\boldsymbol{x}(n)^T\boldsymbol{w}(n+1) = \boldsymbol{x}(n)^T\boldsymbol{w}(n) + \boldsymbol{x}(n)^T\mu e_p(n)x(n) \tag{46}$$

$$\boldsymbol{x}(n)^T\boldsymbol{w}(n+1) = \boldsymbol{x}(n)^T\boldsymbol{w}(n) + \mu e_p(n)\boldsymbol{x}(n)^T x(n) = \boldsymbol{x}(n)^T\boldsymbol{w}(n) + \mu e_p(n)\|\boldsymbol{x}(n)\|^2 \tag{47}$$

If we multiply both sides of equation 47 by -1, and then add to each side $d(n)$, we are able then to substitute the a posterior error $e_p(n)$ into the equation, hence:

$$d(n) - \boldsymbol{x}(n)^T\boldsymbol{w}(n+1) = e_p(n) = d(n) - \boldsymbol{x}(n)^T\boldsymbol{w}(n) - \mu e_p(n)\|\boldsymbol{x}(n)\|^2 \tag{48}$$

We also recognise that the expression $d(n) - \boldsymbol{x}(n)^T\boldsymbol{w}(n)$ is equal to the error $e(n)$ as per the expression in the ordinary LMS algorithm. Therefore:

$$e_p(n) = e(n) - \mu e_p(n)\|\boldsymbol{x}(n)\|^2 \tag{49}$$

which can be rearranged to:

$$e_p(n) = e(n)\left[\frac{1}{1+\mu\|\boldsymbol{x}(n)\|^2}\right] = e(n)\left[1 - \frac{\mu\|\boldsymbol{x}(n)\|^2}{1+\mu\|\boldsymbol{x}(n)\|^2}\right] \tag{50}$$

Now returning to the update rule $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e_p(n)x(n)$ and rearranging to give $\Delta\boldsymbol{w}(n) = \mu e_p(n)x(n)$, we can substitute equation 50 into the update rule to give:

$$\Delta\boldsymbol{w}(n) = \mu e(n)\left[1 - \frac{\mu\|\boldsymbol{x}(n)\|^2}{1+\mu\|\boldsymbol{x}(n)\|^2}\right]\boldsymbol{x}(n) \tag{51}$$

In order to manipulate equation 51 into a form similar to that of the NLMS algorithm provided in the coursework, some rearrangements can be made:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \left[\mu - \frac{\mu^2\|\boldsymbol{x}(n)\|^2}{1+\mu\|\boldsymbol{x}(n)\|^2}\right]e(n)\boldsymbol{x}(n) = \boldsymbol{w}(n) + \left[\frac{\mu}{1+\mu\|\boldsymbol{x}(n)\|^2}\right]e(n)\boldsymbol{x}(n) \tag{52}$$

which can be equivalently written as:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \left[\frac{1}{1+\frac{1}{\mu}\|\boldsymbol{x}(n)\|^2}\right]e(n)\boldsymbol{x}(n) \tag{53}$$

where comparing to the NLMS update provided in the coursework, $\beta = 1$ and $\epsilon = \frac{1}{\mu}$. Therefore, equivalence with the NLMS algorithm is proven.

c) The Generalised Normalised Gradient Descent (GNGD) algorithm is a normalised version of LMS and makes the gain adaptive through the introduction of a regularisation term ϵ. Therefore the cost function is parameterised by ϵ , instead of μ. The effects of regularisation on weight error is evaluated in figure 23, comparing GNGD to Benveniste. For GNGD, $\rho = 0.005$ and $\mu = 1$, while for Benveniste , $\rho = 0.002$ and $\mu = 0.1$, for which it was found through observation to give best performance, and provide stability. As seen, GNGD weights converge faster within 20 samples whilst Benveniste requires 100 samples to achieve steady state. In this instance, the steady state error of Benveniste is marginally lower (note the dB scale), but this is more a reflection of the higher variability in weights for GNGD post-convergence. GNGD additionally benefits for a lower computational complexity. Given that for Benveniste's algorithm includes matrix multiplications when updating sensitivity ψn, for 1) $[\mathbf{I}-\mu(n-1)\mathbf{x}(n-1)\mathbf{x}^{\mathbf{T}}(n-1)]\psi(n-1)$ and within this 2)$\mathbf{x}(n-1)\mathbf{x}^{\mathbf{T}}(n-1)$, these terms result in a complexity of $\mathcal{O}(M^2)$. In constrast, since GNGD only requires vector addition, inner products and scaling procedures, the complexity must be linear of $\mathcal{O}(M)$. Therefore, overall it is the algorithm of choice for convergence and complexity.

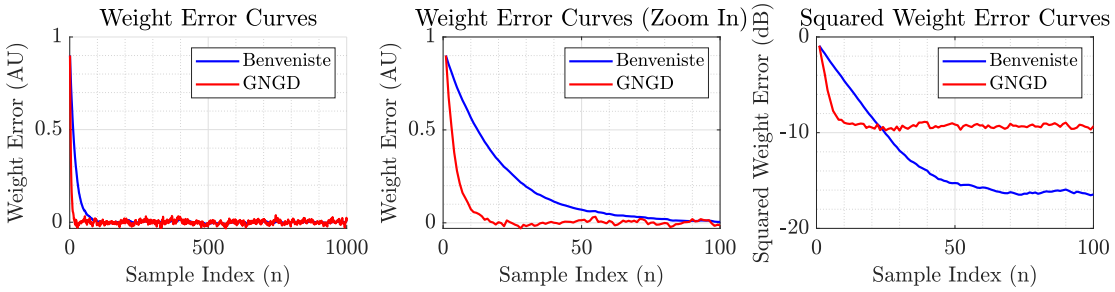


Figure 23: **Left**: GNGD and Benveniste weight error curves. **Middle**: First 100 samples of weight error curves. **Right**: GNGD and Benveniste squared error curves

2.3 Adaptive Noise Cancellation

a) When the Adaptive Line Enhancer (ALE) receives a signal $s(n) = x(n)+\eta(n)$ where $x(n) = sin(0.01n\pi)$ and $\eta(n)$ coloured noise, the ALE delay operator ,Δ, will be selected to decorrelate noise and the clean signal such that the linear predictor component solely removes noise. Following this, a prediction of the clean signal is outputted, $\hat{x}(n)$ and the MSE between predicted and true clean signal defined as:

$$MSE = \mathbb{E}\{(s(n) - \hat{x}_\Delta(n))^2\} = \{(\eta(n) + (x(n) - \hat{x}_\Delta(n)))^2\} \tag{54}$$

$$= \mathbb{E}\{\eta^2(n)\} + \mathbb{E}\{(x(n) - \hat{x}_\Delta(n))^2\} + 2\,\mathbb{E}\{\eta(n)(x(n) - \hat{x}_\Delta(n))\} \tag{55}$$

where the underscore of $\hat{x}_\Delta(n)$ represents the dependency of the ALE prediction on the time shift. From equation (55) we see that only the last term includes 1) a dependency on Δ and 2) inclusion of a noise term. The term $\mathbb{E}[(x(n) - \hat{x}_\Delta(n))^2]$ ideally will equal zero when Δ is appropriately chosen, therefore minimising MSE requires minimisation of the last term only:

$$MSE_{min} = \min_{\Delta}\ \mathbb{E}\{\eta(n)(x(n) - \hat{x}(n,\Delta))\} = \min_{\Delta}\ (\mathbb{E}\{\eta(n)x(n)/ - \mathbb{E}\{\eta(n)\hat{x}(n,\Delta)\}) \tag{56}$$

given that we assume that the clean signal $x(n)$ is uncorrelated with the coloured noise $\eta(n)$, our equation reduces to (sign is not important given that the minimum refers to an absolute value):

$$MSE_{min} = \min_{\Delta}\ \mathbb{E}[\eta(n)\hat{x}(n,\Delta)] = \min_{\Delta}\ \mathbb{E}[(v(n) + 0.5v(n-2))\mathbf{w}^T\mathbf{u}(n,\Delta)] \tag{57}$$

$$= \mathbb{E}\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M-1} w_i s(n-\Delta-i)\} \tag{58}$$

where $\eta(n) = v(n) + 0.5v(n-2)$ and $\hat{x}(n,\Delta) = \boldsymbol{w^T u}(n,\Delta)$, as provided in the documentation. Following this, since the predictor input $\boldsymbol{u}(n,\Delta) = [s(n-\Delta),\ldots,s(n-\Delta-M+1)]^T$, we converted the matrix multiplication to a summation. Expanding upon this:

$$MSE_{min} = \min_{\Delta}\ \mathbb{E}\left\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M-1} w_i(x(n-\Delta-i) + \eta(n-\Delta-i))\right\} \tag{59}$$

$$= \min_{\Delta}\ \mathbb{E}\left\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M-1} w_i(\eta(n-\Delta-i))\right\} \tag{60}$$

$$= \min_{\Delta}\ \mathbb{E}\left\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M-1} w_i\big(v(n-\Delta-i) + 0.5v(n-2-\Delta-i)\big)\right\} \tag{61}$$

$$= \min_{\Delta}\ \sum_{i=0}^{M-1} w_i E\left\{(v(n) + 0.5v(n-2))\big(v(n-\Delta-i) + 0.5v(n-\Delta-2-i)\big)\right\} \tag{62}$$

where the term $x(n-\Delta-i)$ was removed from equation (61) as it is uncorrelated with $\eta(n) = u(n) + 0.5u(n-2)$. Concluding from equation (62), since $u(n)$ is identically and independently distributed white noise, it holds true that $\mathbb{E}\{u(n)u(n-m)\} = 0$, meaning that if equation 62 is to be minimised to equal 0, the condition must in turn hold that $\Delta > 2$ meaning the minimum possible value is 3. To justify this minimum, ALE was implemented using the LMS algorithm and applied to 100 iterations of $s(n)$, fixing the filter order ,$M = 5$ 4, and the learning rate $\mu = 0.01$. As shown in figure 24, when transitioning from $\Delta = 2$ to $\Delta = 3$, the MSPE metric indicating ALE-LMS performance experiences a significant decrease, by 0.16.

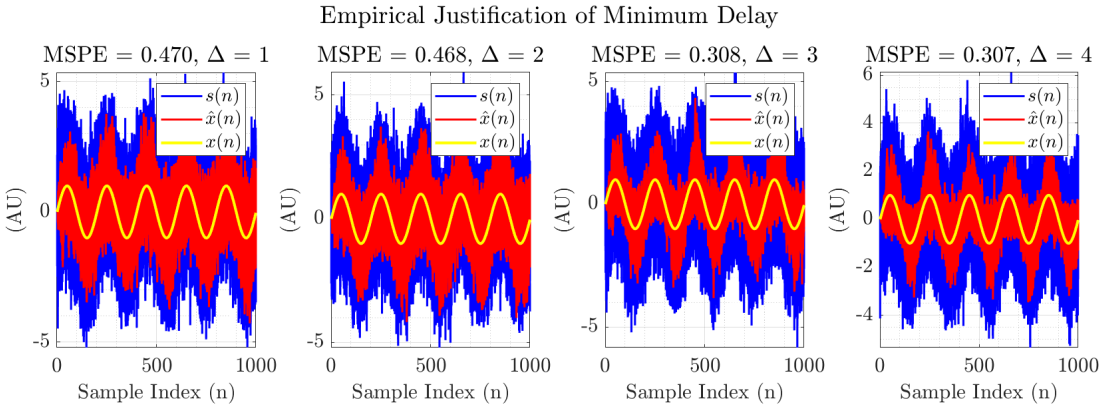


Figure 24: Investigating the effect of delay Δ on Minimum Squared Prediction Error on the performance of the ALE-LMS, for a fixed filter order M =5.

b) Following from above, both the delay, Δ, and the filter order ,M, are varied with values $\Delta = [1:25]$, $M = [5, 10, 15, 20]$ to investigate their effects on MSPE and plotted in figure 25. Firstly, increasing the filter order results in increases in MSPE acros all delay values. MSPE curves for all filters have the same shape 1) when $\Delta \leq 2$ the MSPE is higher and decreases reaching minima at $\Delta = 3$ or $\Delta = 4$ ($M = 5 \rightarrow \Delta = 3$, $M = 10 \rightarrow \Delta = 4$, $M = 15 \rightarrow \Delta = 4$, $M = 20 \rightarrow \Delta = 4$). Note however for $3 \leq \Delta \leq 6$, MSPE values are only marginally different, and so more iterations may be required to give more confidence in these minima. For $\Delta > 6$, there is a monotonic increase in MSPE with increasing delay due to a significant phase shift in the predicted signal $\hat{x}(n)$ relative to the input $x(n)$. When Δ was fixed at 3, the relationship between filter order and MSPE was further explored, depicted in figure 27, for $M = [1:20]$. A minima is reached at $M = 3$, while MSPE monotonically increases for $M > 3$, as increasing model complexity results in additional degrees of freedom that will be overfitted to noise, reducing prediction performance. Overall a combination of $M = 3$ and $\Delta = 3$ (or $M = 5$ when selecting from $M = [5, 10, 15, 20]$) is an optimal choice for parameterisation, given that minimising model order is pragmatic in turns of minimising computational complexity.

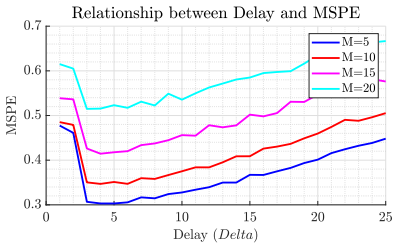


Figure 25: Relationship between delay, Δ, and MSPE for different filter orders, M.

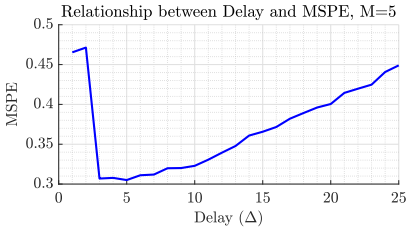


Figure 26: Relationship between delay, Δ, and MSPE for filter order, M = 5.

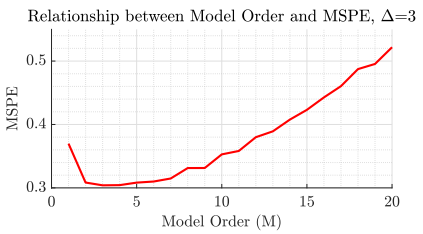


Figure 27: Relationship between filter order, M, and MSPE for delay $\Delta = 3$.

c) The Adaptive Noise Cancellation configuration (ANC) was implemented in MATLAB and the performance compared to ALE via the MSPE metric. As part of ANC, a secondary noise input $\epsilon(n)$ was be generated that is correlated with the primary noise input $\eta(n)$. With this, ANC estimates thee primary

noise, $\hat{\eta}(n)$ through matrix multiplication of the sequence of $\epsilon(n)$ values with the filter weights, which is in turn used to estimate the clean signal as $\hat{x}(n) = s(n) - \hat{\eta}(n)$. For this stimulation, $\epsilon(n) = 1.2\eta(n) + 0.1$ while $\Delta = 3$ (ALE only), $M = 5$ and $\mu = 0.01$ and 100 iterations generated. As shown in figure 28, the MSPE for ANC is approximately $\frac{1}{6}$ that for ALE demonstrating that ANC has significantly improved capability when denoising the input signal $s(n)$. However there are caveats associated with superior ANC performance: 1) MSPE is highly dependent upon tactical selection of ϵ which may not always be possible; for instance if the constant 0.1 is increased to 0.5 it significantly decreases correlation between ϵ and η thereby decreasing performance; and 2) ANC suffers from a long convergence time where in this case it takes 1/4 of the signal length (although is dependent on coefficients defining ϵ's correlation with $\eta(n)$), therefore ALE may be a more practical choice if the user is handling signals of low sample lengths. Finally, the ensemble of the 100 simulated realisations was plotted to showcase the superior predictive capabilities of ANC.

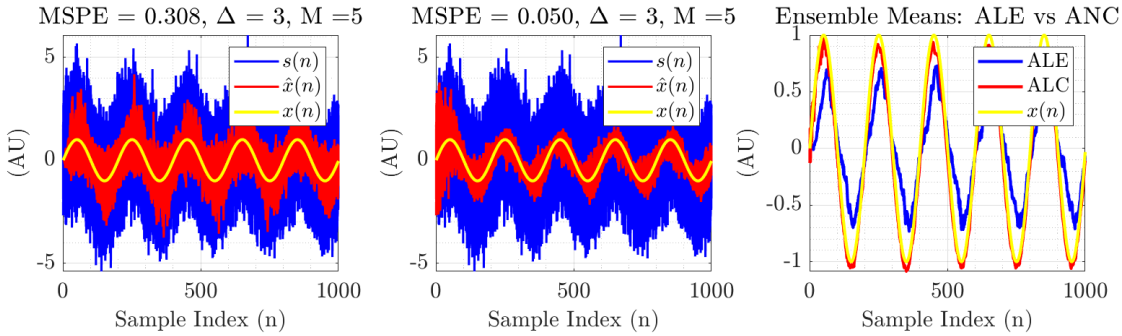


Figure 28: **Left**: 100 realisations of input signal $s(n)$, ALE-predicted clean signal $\hat{x}(n)$ and clean signal $x(n)$. **Middle**: 100 realisations of input signal $s(n)$, ANC-predicted clean signal $\hat{x}(n)$ and clean signal $x(n)$. **Right**: Ensemble averages of ALE and ANC clean signal estimates, and the true clean signal.

d) Now using real-world data, ANC was applied to the Cz electrode of an EEG recording, with the aim to remove mains noise (50Hz) without incurring information loss in the remaining frequency bands. To implement ANC, the secondary noise input was set as a unit-amplitude 50Hz sinusoid, noise-corrupted by white Gaussian noise with noise power 0.005. For the spectrogram, a Hanning window of length 4096 samples was used with a overlap of $\frac{2}{3}$ between windows, and an FFT size for each frame of Cz data of $5 \times 4096 = 20480$. The spectrogram of the mains noise-corrupted EEG signal is shown in figure 29.

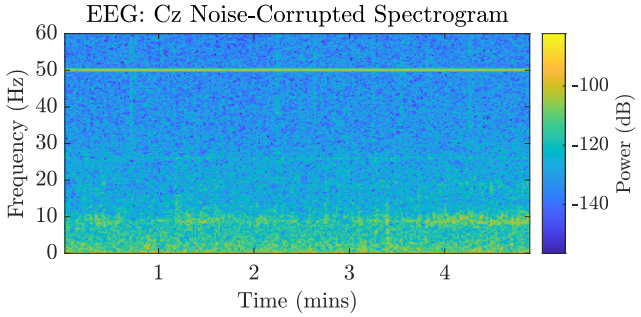


Figure 29: Noise-corrupted spectrogram of an EEG signal.

When applying ANC for de-noising, the influence of filter order, M, and learning rate μ on removal of mains noise and effects on surrounding frequencies was explored, shown in figure 30. Increasing μ increases the effects of ANC de-noising on surrounding frequency bands given larger jumps in filter weights. Increasing M also increases de-noising effects but when too high ($M = 30$), the power in the 50Hz band is too low relative to surrounding bands, leaving an artifact in the spectrogram. In consideration of this, using $M = 10$ and $\mu = 0.001$ is the optimal choice given that adequate 50Hz-related power is removed to prevent artifacts and distortion of neighbouring bands. With these parameters, the periodogram of noise-corrupted Cz data and post-ANC data is plotted in figure 31 alongside an error periodogoram (for which

an absolute difference between the pre-ANC and post-ANC power density is computed), showcasing that the greatest difference in power pre and post ANC is at 50Hz. Note that the periodogram was constructed using Welchs's method, with window size of 1024 samples and FFT size $5 \times 1024 = 5120$.

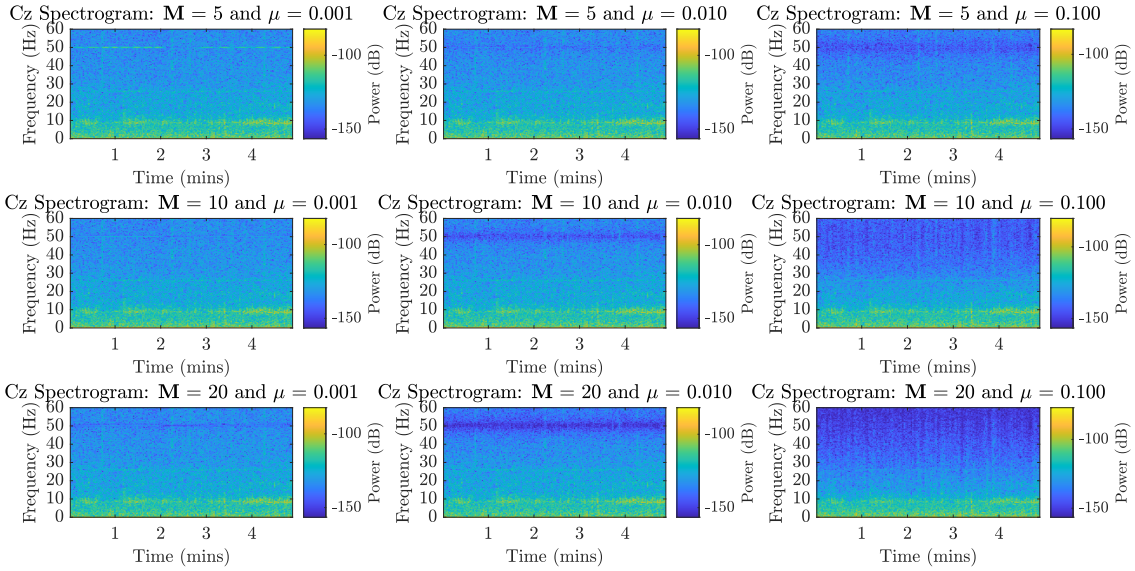


Figure 30: Spectrograms of post-ANC denoising of an EEG signal, across model orders $M = [5, 10, 20]$ and learning rates $\mu = [0.001, 0.01, 0.1]$.

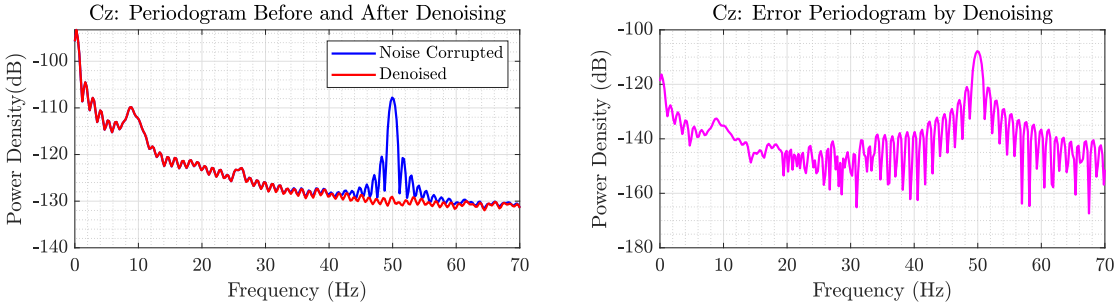


Figure 31: **Left**: Pre-ANC and post-ANC EEG electrode Cz periodograms, constructed via Welch's method.**Right**: Absolute error periodogram produced from difference in the power densities of pre and post-ANC EEG data.

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

a) A widely-linear-moving-average first-order process, WLMA(1), driven by circular WGN, was generated in MATLAB with the following formulation:

$$y(n) = x(n) + b_1 x(n-1) + b_2{}^*(n-1) \qquad x \sim \mathcal{N}(0,1) \tag{63}$$

The real and imaginary components of the signal and the WGN driver are shown on the left of figure 32. Plotting these components verifies that WGN is indeed a circular process, for which the underlying probability distribution is rotation invariant. In contrast, given the non-circular depiction of $y(n)$ when plotted, we confirm that the signal is non-circular. Consequently, we can predict that using complex LMS (CLMS) as a candidate estimation algorithm for processing our complex-valued signal will result in greater

steady-state error during algorithm learning than when using augmented complex LMS (ACLMS). This is because CLMS fails to correctly account for second-order statistical relations, particularly concerning pseudocovariance. For circular data, the pseudocovariance matrix is all-zeros, given that the powers in the real and imaginary channels are equal (${\sigma_{Re}}^2 = {\sigma_{Im}}^2$) such they are uncorrelated. Therefore they are 'proper', where 'properness' is the second-order statistical property. In the case of $y(n)$, the pseudocovariance matrix is not all-zeros and as such, it is an 'improper' signal. ACLMS can appropiately account for data improperness due to the additional degrees of freedom it possesses, hence is predicted to result in less error in prediction. Based on the results on the right of figure 32, we see as expected that the steady state error from ACLMS (with the metric defined as $10\log 10(|e(n)|)^2$, in dB) is 3.89 (3.s.f) times lower than CLMS steady state error, given the inability of CLMS to correctly account for second-order statistics.

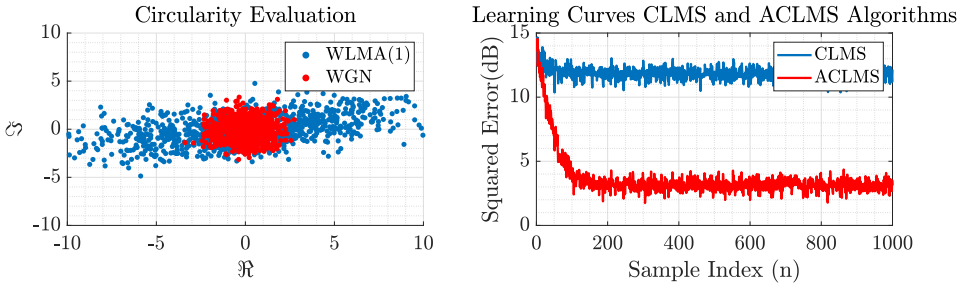


Figure 32: **Left**: Circularity plots of WGN and signal $y(n)$. **Right**:Learning curves evalauting prediction perfromance of CLMS and ACLMS for signal $y(n)$.

As a final note, we can use the pseudocovariance/properness property to quantify the degree of circularity of data ρ - the ratio of pseduocovariance to covariance. Hence, the lower the value (in the range [0,1]) the more circular the data. For WGN, $\rho = 0.00$ (3.s.f) and for $y(n)$, $\rho = 0.85$ (3.s.f), corroborating the discussions above.

b) Complex-valued signals $v(n)$ were generated using data corresponding to low, medium and high wind speeds, where $v[n] = v_{east}[n] + jv_{north}[n]$, and their circularity plots shown in figure 33. The circularity coefficients were computed (as described previously) and found to equal : $\rho_{low} = 0.159$, $\rho_{medium} = 0.454$ and $\rho_{high} = 0.624$ (3.d.p). Hence, with increasing speed circularity decreases. Therefore, it would be expected that in all cases ACLMS should give a predicted signal that leads to less prediction error than CLMS, and for the difference in performance between ACLMS and CLMS to increase with speed. The superior performance of ACLMS across all wind speed data, as measured by the Minimum Prediction Squared Error, is confirmed with figure 34. Moreover, the greatest improvement in performance from CLMS to ACLMS is seen for the high-wind-speed data where there is a 0.1dB improvement in performance (based on the minima of the CLMS and ACLMS MSPE curves). We also see that the lowest MPSE values of ACLMS are achieved at lower filter orders than the lowest MPSE values of CLMS, meaning that ACLMS offers the additional benefit of lower model complexity. The minimum MPSE values of ACLMS occur at orders [4,5,4] and for CLMS at [7,9,5], for low, medium and high-wind-speeds respectively.

However, as the filter order increases above 9,9 and 23 for low,medium and high-wind-speed-data respectively , the performance of ACLMS becomes worse than CLMS. Over-parameterising the model results in overfitting to noise, and this affects ACLMS more greatly given it already has more degrees of freedom to accommodate for non-circularity. This explains why the cross-over point when ACLMS performance becomes worse than CLMS increases with wind-speed, as high-wind-speed data has a larger circularity coefficient, so better suits the additional degrees of freedom of ACLMS. At the lowest filter orders (≤ 3) , both ACLMS and CLMS see increases in error due to underfitting to the input data. Finally, for both ACLMS and CLMS, the absolute MPSE across all filter orders decreases with wind speed - meaning the performance decreases - suggesting that both methods are still negatively affected by increasing non-circularity.

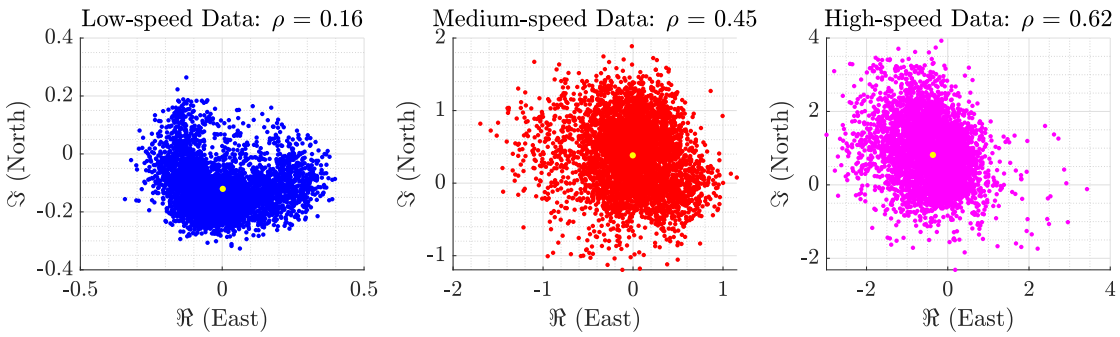


Figure 33: Circularity plots of low-speed (blue), medium-speed (red) and high-speed (pink) wind data. Yellow dots refer to the mean.

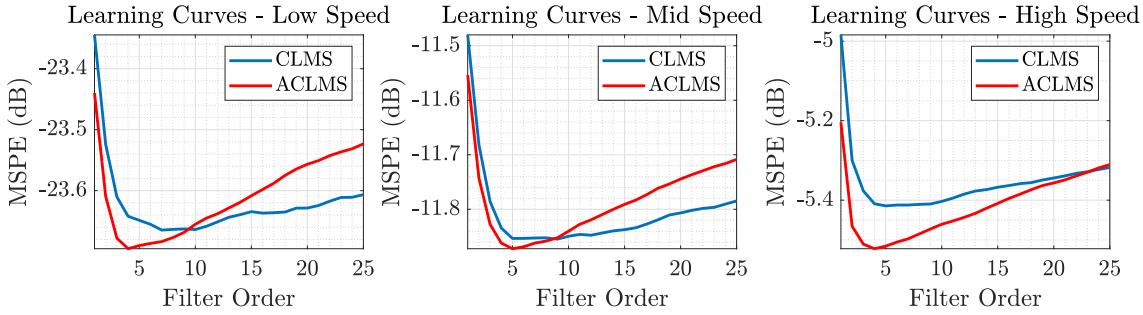


Figure 34: CLMS and ACLMS learning curves for low,medium and high speed data. Metric of performance is Minimum Prediction Squared Error (MPSE). Note that the learning rate μ differed across simulations to avoid destabilisation at different speed data: $\mu_{low} = 0.001$, $\mu_{mid} = 0.005$,$\mu_{high} = 0.1$.

c) Balanced and unbalanced sets of complex voltages were generated using the Clarke Transform. The Clarke Transform projects the three-phase voltages of a three phase power system, v_a, v_b, v_c onto two orthogonal axes to form the I and Q components, namely $v_\alpha(n)$ and $v_\beta(n)$, in turn combined to form the complex Clarke voltage $v(n) = v_\alpha(n) + jv_\beta(n)$. When generating the balanced complex voltage 1) the magnitudes of the three-phase voltages equal one another: $V_a = V_b = V_c$, and 2) the phase shift between the three-phase voltages is exacrlt 120^o :$\Delta_b = \Delta_c = 0$. If either or both conditions are violated, the complex voltage is unbalanced. Based on this, balanced and unbalanced voltages were simulated and plotted below in figures 35 and 36. Phase and magnitude parameters are specificed on these figures.

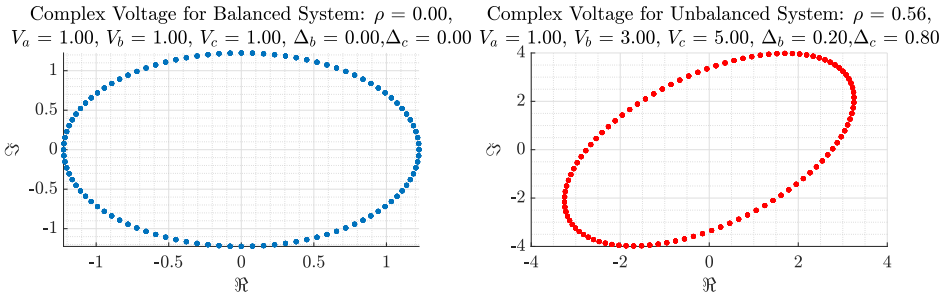


Figure 35: **Left**: Complex voltage circularity plot for example balanced system. **Right**: Complex voltage circularity plot for example unbalanced system, where both magnitude and phase conditions are violated.

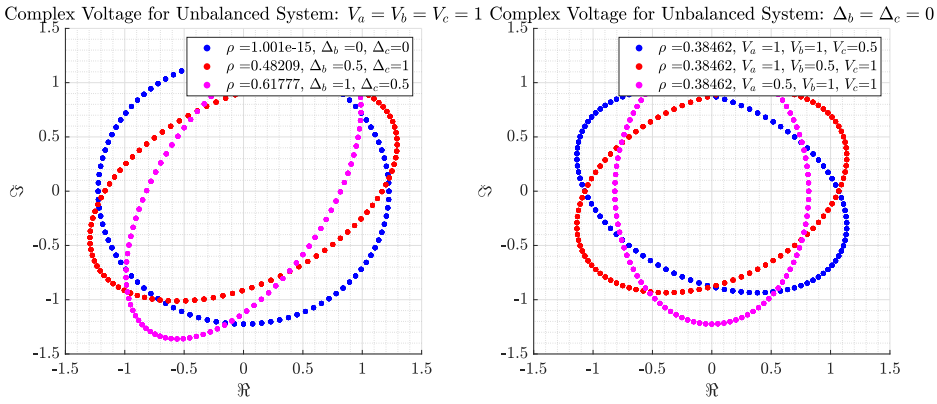


Figure 36: **Left**: Complex voltage circularity plots for unbalanced systems where only phase condition is violated. **Right**: Complex voltage circularity plots for unbalanced systems where only magnitude condition is violated.

These simulations demonstrate that balanced systems produce circular complex voltages and therefore $|\rho| = 0$, while unbalanced systems produce non-circular complex voltages and therefore $0 < |\rho| \leq 1$. More specifically shown in figure 36, if only the magnitude condition is violated (right figure), sequential modification of one voltage amplitude results in 'sagging' of the circularity plot, in a direction consistent with the phase shift of the voltage affected e.g. when $V_c = 0.5$, and v_c has a phase shift of $+\frac{2\pi}{3}$, the stretching gives the appearance of an oval rotated by $+\frac{2\pi}{3}$ about the vertical axis clockwise. In contrast, for equal modifications in Δ_b and Δ_c whilst magnitudes are fixed, the circularity changes differently, given that these are combined with already existing phase shifts, thereby weighting the voltage magnitudes differently. Overall however, it is clear that such plots could be used to accurately and quickly identify system faults. For instance, if there was a short-circuit resulting in one of the three-phase voltage significantly dropping, this would reflect in the degree of non-circularity of the complex Clarke voltage. Moreover, it can aid in deciphering if the fault affects one or more voltages, and the fault severity.

d) We are provided with:

1. A **strictly linear** AR(1) process: $v(n+1) = h^*(n)v(n)$
2. A **widely linear** AR(1) process: $v(n+1) = h^*(n)v(n) + g^*(n)v^*(n)$

where h and g are coefficient vectors. It is given that under the Clarke transform, the complex Clarke voltage consists of two components α and β. Any given balanced complex Clarke voltage $v(n)$ is represented as the complex exponential:

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n + \phi)} \tag{64}$$

where f_s is the sampling frequency, f_o the frequency of the balanced complex system/process, n the discrete time index and ϕ a phase shift. If we use equation (64) to represent the complex voltage at the next time step:

$$v(n+1) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s}(n+1)+\phi)} = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n+\phi)} e^{j2\pi \frac{f_o}{f_s}} = v(n) e^{j2\pi \frac{f_o}{f_s}} \tag{65}$$

Following this, given that the strictly linearly AR(1) model is described as $v(n+1) = h^*(n)v(n)$, then:

$$h^*(n) = e^{j2\pi \frac{f_o}{f_s}} \tag{66}$$

$$h(n) = e^{-j2\pi \frac{f_o}{f_s}} \tag{67}$$

Using Euler's form, we can express $h(n)$ in terms of a magnitude, and angle (i.e. the arctan of the

imaginary and real components of $h(n)$), which gives the equation:

$$h(n) = e^{-j2\pi\frac{f_o}{f_s}} = |h(n)|e^{j\left(arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right)\right)} \tag{68}$$

Hence given the equivalency in the arguments of the exponential terms:

$$f_o = \frac{f_s}{2\pi} arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \tag{69}$$

noting that the minus sign in equation (68) represents a rotation of the signal about the complex axis ,so does not impact interpretation of f_o. Therefore, the frequency of the balanced complex voltage is derived, using the strictly linear process equation.To then derive the frequency of the unbalanced complex Clarke voltage, the widely linear process equation is used, as the additional degree of freedom permits representation of non-circularity. Any given unbalanced complex Clarke voltage $v(n)$ is represented as:

$$v(n) = A(n)e^{j(2\pi\frac{f_o}{f_s}n+\phi)} + B(n)e^{-j(2\pi\frac{f_o}{f_s}n+\phi)} \tag{70}$$

with the same definitions of the parameters as in the representation of the balanced complex Clarke voltage. Additionally, coefficients $A(n)$ and $B(n)$ are expressions parameterised by the peak voltage values of a three-phase power system. At time index $n+1$:

$$v(n+1) = A(n+1)e^{j(2\pi\frac{f_o}{f_s}(n+1)+\phi)} + B(n+1)e^{-j(2\pi\frac{f_o}{f_s}(n+1)+\phi)} \tag{71}$$

we can also substitute equation (70) into our definition of the widely linear AR(1) process, $v(n+1) = h^*(n)v(n) + g^*(n)v^*(n)$,to give an additional expression for $v(n+1)$:

$$h^*(n)\left[A(n)e^{j(2\pi\frac{f_o}{f_s}n+\phi)} + B(n)e^{-j(2\pi\frac{f_o}{f_s}n+\phi)}\right] + g^*(n)\left[A^*(n)e^{-j(2\pi\frac{f_o}{f_s}n+\phi)} + B^*(n)e^{j(2\pi\frac{f_o}{f_s}n+\phi)}\right] \tag{72}$$

comparing the terms of equations (71) and (72):

$$v(n+1) = A(n+1)e^{j(2\pi\frac{f_o}{f_s}(n+1)+\phi)} + B(n+1)e^{-j(2\pi\frac{f_o}{f_s}(n+1)+\phi)} \tag{73}$$

$$= A(n+1)e^{j(2\pi\frac{f_o}{f_s}n+\phi)}e^{j(2\pi\frac{f_o}{f_s})} + B(n+1)e^{-j(2\pi\frac{f_o}{f_s}n+\phi)}e^{-j(2\pi\frac{f_o}{f_s})} \tag{74}$$

$$= h^*(n)A(n)e^{j(2\pi\frac{f_o}{f_s}n+\phi)} + h^*(n)B(n)e^{-j(2\pi\frac{f_o}{f_s}n+\phi)} + g^*(n)A^*(n)e^{-j(2\pi\frac{f_o}{f_s}n+\phi)} + g^*(n)B^*(n)e^{j(2\pi\frac{f_o}{f_s}n+\phi)} \tag{75}$$

hence with the presented equivalency, we can deduce expressions for $e^{j(2\pi\frac{f_o}{f_s})}$ and $e^{-j(2\pi\frac{f_o}{f_s})}$ as:

$$e^{j2\pi\frac{f_o}{f_s}} = \frac{h^*(n)A(n) + g^*(n)B^*(n)}{A(n+1)} \tag{76}$$

$$e^{-j2\pi\frac{f_o}{f_s}} = \frac{h^*(n)B(n) + g^*(n)A^*(n)}{B(n+1)} \tag{77}$$

if we assume that $A(n+1) \approx A(n)$ and $B(n+1) \approx B(n)$, we can approximate that:

$$e^{j2\pi\frac{f_o}{f_s}} \approx h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \tag{78}$$

$$e^{-j2\pi\frac{f_o}{f_s}} \approx h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \tag{79}$$

since $e^{j2\pi\frac{f_o}{f_s}}$ and $e^{-j2\pi\frac{f_o}{f_s}}$ are complex conjugates of one another, if we take the complex conjugate of

equation (79), it can be equated to equation (78) to give:

$$h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} = h(n) + g(n)\frac{A(n)}{B^*(n)} \tag{80}$$

let $X(n) = \frac{B^*(n)}{A(n)}$

$$h^*(n) + g^*(n)X(n) = h(n) + g(n)\frac{1}{X(n)} \tag{81}$$

equation (81) can be rearranged to give a quadratic expression:

$$g^*(n)X^2(n) + (h^*(n) - h(n))X(n) - g(n) = 0 \tag{82}$$

solving equation (82) for $X(n)$ gives:

$$X(n) = \frac{-(h^*(n) - h(n)) \pm \sqrt{(h^*(n) - h(n))^2 + 4g^*(n)g(n)}}{2g^*(n)} \tag{83}$$

given that for a complex number, such as $h(n)$, $h(n) + h^*(n) = 2\Re\{h(n)\}$ and $h(n) - h^*(n) = 2\Im\{h(n)\}j$

$$X(n) = \frac{2\Im\{h(n)\}j \pm j\sqrt{-4\Im\{h(n)\}^2 + 4|g(n)|^2}}{2g^*(n)} = \frac{\Im\{h(n)\}j \pm j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2}}{g^*(n)} \tag{84}$$

this expression for $X(n)$ (and hence $\frac{B^*(n)}{A(n)}$) can be substituted into equation (78) to give:

$$e^{j2\pi\frac{f_o}{f_s}} = h^*(n) + \Im\{h(n)\}j \pm j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2} = \Re\{h(n)\} \pm j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2} \tag{85}$$

Only one solution is needed for f_0, choosing the positive solution for ease of notation (the sign again is merely representative of a rotation). Expressing equation (85) in terms of Euler notation, with sign in mind:

$$e^{j2\pi\frac{f_o}{f_s}} = Ye^{j\left(\frac{\sqrt{\Im\{h(n)\}^2 - |g(n)|^2}}{\Re\{h(n)\}}\right)} \tag{86}$$

where Y is a constant, that must be greater than 0. Finally, since the equivalence between the arguments to the exponentials on either side of the equation must hold, we equate them and solve for f_0:

$$f_o = \frac{f_s}{2\pi} arctan\left(\frac{\sqrt{\Im\{h(n)\}^2 - |g(n)|^2}}{\Re\{h(n)\}}\right) \tag{87}$$

hence, the frequency of the unbalanced complex Clarke voltage is derived by using in this instance, the equation of the widely linear AR(1) process.

e) First order CLMS and ACLMS were used for the prediction of the nominal frequency (set here to UK power system frequency 50Hz) for both balanced and unbalanced systems. The magnitude and phase parameters for these simulations are equal to those found in figure 35, while the learning rate $\mu = 0.02$ and sampling frequency $f_s = 1000$. Additionally, given equations (69) and (87) have their denominator equal to $\Re\{h(n)\}$, h(0) is set to equal 1 to avoid destabilising causing overshoot in the estimate of f_o.

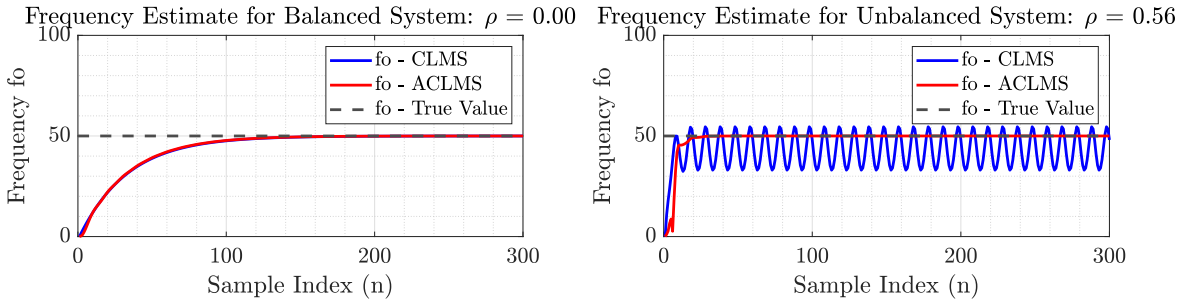


Figure 37: Estimation of nominal frequency $f_o = 50Hz$ for a balanced and unbalanced system, $f_s = 1000Hz$, $\mu = 0.02$.

As shown in figure 37, both CLMS and ACLMS algorithms can estimate the true nominal frequency value, converging to the true value after ~ 150 samples. However, only ACLMS can correctly estimate the true nominal frequency value f_o, whilst the estimate from CLMS oscillates about a value lower than f_o, oscillating about a mean of 43.3Hz (3.s.f). This result is expected given that ACLMS has an additional degree of freedom ($g(n)$) to account for. The bias is expected because the generalised form of the complex voltage for the three-phase power system is $v(n) = A(n)e^{n(\omega n\Delta T+\phi)} + B(n)e^{-j\omega n\Delta T+\phi}$ where $B(n) = 0$ for balanced systems. Therefore, if the unbalanced system uses both terms and the balanced system can only learn the real term, bias will inevitably arise. The inferior performance of the balanced system with respect to oscillation i.e. large error variance, can be mathematically justified when evaluating the complex voltage steady state. The generalised unbalanced complex voltage form remains as the expression for $v(k)$, meaning the one-step ahead estimation by the strictly linear CLMS algorithm is:

$$\hat{v}(n+1) = (A(n)e^{j(\omega n\Delta T+\phi)} + B(n)e^{-j\omega n\Delta T+\phi})h(n) \tag{88}$$

Upon achieving steady steady state, $\hat{v}(n+1) \approx v(n+1)$ such that:

$$h(n) = \frac{A(n+1)e^{j(\omega n\Delta T+\phi)}e^{j\omega\Delta T}}{(A(n)e^{j(\omega n\Delta T+\phi)} + B(n)e^{-j\omega n\Delta T+\phi})} + \frac{B(n+1)e^{-j(\omega n\Delta T+\phi)}e^{-j\omega\Delta T}}{(A(n)e^{j(\omega n\Delta T+\phi)} + B(n)e^{-j\omega n\Delta T+\phi})} \tag{89}$$

Under the approximations 1) $A(n+1) \approx A(k)$ and 2) $B(n+1) \approx B(n)$ and 3) sampling frequency is much greater than nominal system frequency:

$$h(n) = e^{-j\omega\Delta T} + \frac{e^{j\omega\Delta T} - e^{-j\omega\Delta T}}{1 + \frac{B(n)}{A(n)}e^{-2j(\omega n\Delta T+\phi)}} \tag{90}$$

for which $h(n) = h(n + (\frac{1}{2f\Delta T}))$ is a periodic variable, and $\frac{B(n)}{A(n)}$ is unknown. Given the deduction of nominal frequency ,f_o, we know it contains the monotonically increasing function *arctan*, with input argument $h(n)$. Hence, when $h(n)$ is periodic, so too will be the output of *arctan* and in turn the CLMS estimate of f_o ,when it undermodels an unbalanced power system. The oscillation frequency is $2f_o$, as depicted in the simulation given $f_s = 1000Hz$. However, when $B(k) = 0$ in a balanced system, $h(k)$ is no longer periodic and so oscillations do not arise.

3.2 Adaptive AR Model Based Time-Frequency Estimation

a) A 1500-sample frequency modulated (FM) signal $y(n) = e^{j(\frac{2\pi}{f_s}\phi(n))} + \eta(n)$, where noise driver $\eta(n)$ represents zero-mean circular complex-valued white noise with ${\sigma_\eta}^2 = 0.05$, was generated and plotted in figure 38. Note that the phase (right) plot is generated by $\int f(n)\,dn$, where f(n) is also depicted (left). From this it is clear that our signal is non-stationary due to the time-variance in frequency.

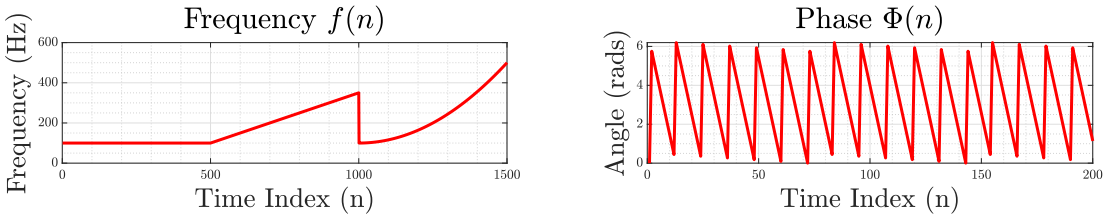


Figure 38: Estimation of nominal frequency $f_o = 50Hz$ for a balanced and unbalanced system, $f_s = 1000Hz$, $\mu = 0.02$.

Following this, AR coefficients for orders 1,5 and 10 were derived using `aryule`, the power spectra obtained with `freqz` and plotted in figure 39. As expected irrespective of AR model order, AR models fail to accurately capture all frequency information, give that `aryule` assumes signal stationarity. Even for AR(10), whilst there is a spectral peak at 100Hz representing the frequency within $[1, 500]$, the latter two peaks (at 250 and 750 Hz) do not relate to the frequency behaviour within $(500, 1500]$. In contrast, AR(1) peaks at 177.7Hz (1.d.p) which is the average frequency of our FM signal.

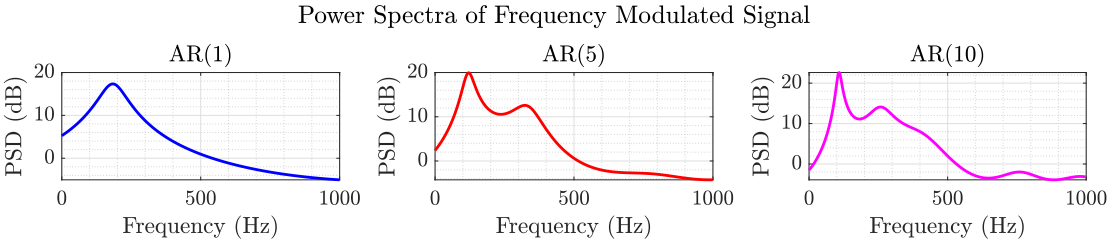


Figure 39: Power spectra ofAR(1), AR(5) and AR(10) models of a frequency modulated signal.

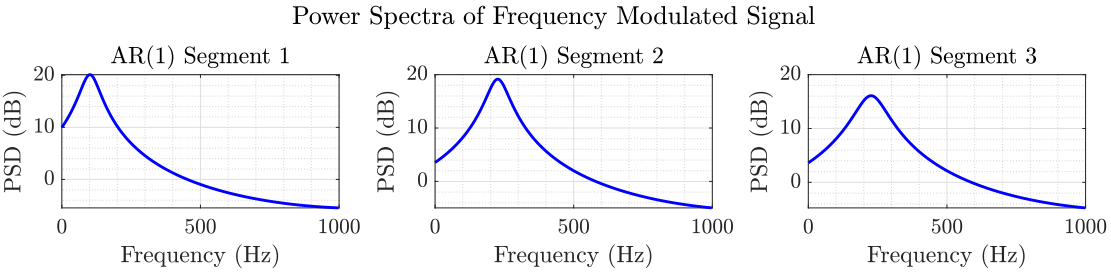


Figure 40: Piecewise AR(1) modelling of a frequency modulated signal.

Given that the frequency is a branch function, we re-evaluate the performance of the AR(1) model splitting the model into it's three different segments, defined by function behaviour 1) constant frequency ($[1, 500]$), 2) linear frequency function ($(500, 1000]$) and 3) quadratic frequency function ($(1000, 1500]$). As shown in figure 40, the model still fails at representing frequency information in segments 2 and 3, but can represent the 100Hz frequency of segment 1. This is since segments 2 and 3 are non-stationary frequency functions, whilst segment 1 is stationary. A final improvement could be to window the function into smaller segments e.g. 5-length-samples, and use the AR(1) model to estimate the frequency within each window in a piecewise manner (under the assumption that the frequency within a window is approximately constant).

b) Since the AR model assumes stationarity , we instead test the performance of the CLMS algorithm in estimating the AR(1) coefficient and in turn, to derive the time-frequency characteristics of the signal. This algorithm is more appropriate because 1) it is an adaptive approach that does not assume stationarity i.e. unlike `aryule` it does not use all signal samples at once, but instead uses past values across iterations to adapt filter coefficients over time, and 2) because our signal is almost perfectly circular ($|\rho| = 0.018$). As seen in figure 41, CLMS is able to capture time-frequency information that is time-variant, but is

dependent on the learning rate μ. When μ is too small e.g. 0.005, the filter coefficients do not converge before using all signal samples and so cannot adapt in time to varying frequency content. However, when μ is too large e.g. 0.5, oscillations arise in the time-frequency spectrum due to the large steps in learning, increasing the variance in the estimate.

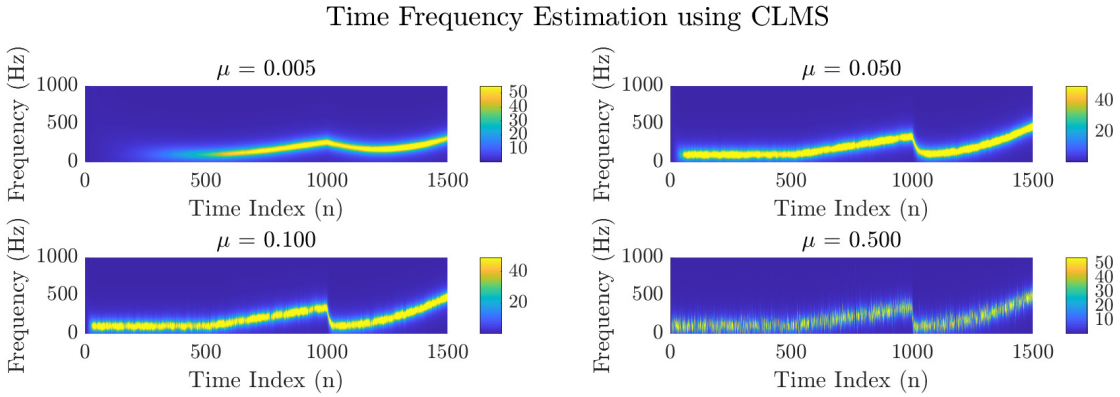


Figure 41: CLMS with different step-sizes μ performed on a frequency modulated signal.

3.3 A Real Time Spectrum Analyser Using Least Mean Square

a) As given, a signal $y(n)$ is to be estimated with a linear combination of N harmonically related sinusoids, such that the estimate $\hat{y}(n)$ can be represented as:

$$\hat{y(n)} = w(k) \sum_{k=0}^{N-1} e^{\frac{j2\pi kn}{N}} \tag{91}$$

where the matrix containing all estimates of $y(n)$ is expressed as:

$$\begin{bmatrix} \hat{y}(0) \\ \hat{y}(1) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{\frac{j2\pi(1)(1)}{N}} & \dots & e^{\frac{j2\pi(1)(N-1)}{N}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{\frac{j2\pi(N-1)(1)}{N}} & \dots & e^{\frac{j2\pi(N-1)(N-1)}{N}} \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ \vdots \\ w(N-1) \end{bmatrix} = \boldsymbol{Fw} \tag{92}$$

This is also equivalent to $\hat{\boldsymbol{y}} = \boldsymbol{Fw}$, for which y is the vector of estimates, $\boldsymbol{F}$ the matrix of harmonically related sinusoids, and $\boldsymbol{w}$ the vector of unknown weights (Fourier coefficients). The least squares problem is solved with a cost function, which is the L_2-norm of the prediction error:

$$\mathcal{J}(\boldsymbol{w}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 = \|\boldsymbol{y} - \boldsymbol{Fw}\|^2 = (\boldsymbol{y} - \boldsymbol{Fw})^H(\boldsymbol{y} - \boldsymbol{Fw}) \tag{93}$$

To deduce the least squares solution, the Fourier coefficients must minimise the cost function. Therefore:

$$\frac{\partial \mathcal{J}(\boldsymbol{w})}{\partial \boldsymbol{w}}\bigg|_{w=w*} = \frac{\partial(\boldsymbol{y^H y} - \boldsymbol{y^H Fw} - \boldsymbol{w^H F^H y} + \boldsymbol{w^H F^H Fw})}{\partial \boldsymbol{w}}\bigg|_{w=w*} = 0 \tag{94}$$

where the numerator of the partial differential is the expansion of equation (93). Upon differentiation:

$$\frac{\partial \mathcal{J}(\boldsymbol{w})}{\partial \boldsymbol{w}}\bigg|_{w=w*} = 0 - \boldsymbol{F^H y} - \boldsymbol{F^H y} + 2\boldsymbol{F^H Fw} = 2\boldsymbol{F^H Fw} - 2\boldsymbol{F^H y} = 0 \tag{95}$$

We can now rearrange equation (95) for the optimum Fourier coefficients $\boldsymbol{w^*}$:

$$\boldsymbol{w^*} = (\boldsymbol{F^H F})^{-1}\boldsymbol{F^H y} \tag{96}$$

Therefore, it is clear that for equation (96) to be a valid solution, the matrix $\boldsymbol{F}$, along with the expression $\boldsymbol{F^H F)^{-1}}$ must be invertible. In this instance, $\boldsymbol{F}$ is invertible because the matrix is full-rank, based on upon the intrinsic orthogonality of basis vectors i.e. each column of $\boldsymbol{F}$ represents a sinusoid with a different frequency. Exploiting this, equation (96) can be simplified knowing that $\boldsymbol{F^H F} = N\boldsymbol{I}$. Hence:

$$\boldsymbol{w^*} = (\boldsymbol{F^H F})^{-1}\boldsymbol{F^H y} = (N\boldsymbol{I})^{-1}\boldsymbol{F^H y} = \frac{1}{N}\boldsymbol{F^H y} \tag{97}$$

Now referring to the Discrete Fourier Transform, the transform is used to convert a finite sequence in the time domain to a sequence in the frequency domain. Conversely, the Inverse Discrete Fourier Transform performs conversion from frequency to the time domain. These are respectively described as:

$$\boldsymbol{DFT:} \qquad X(k) = \frac{1}{N}\sum_{k=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nk} \tag{98}$$

$$\boldsymbol{IDFT:} \qquad \hat{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}nk}) \tag{99}$$

By comparing equation (99) with equation (91), and noting the matrix form of $\hat{y}(n)$ as $\boldsymbol{y} = \boldsymbol{Fw}$, we can see that there is similarity, where:

$$\hat{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi}{N}nk} = \boldsymbol{F_{DFT}X} = \frac{1}{N}\boldsymbol{FX} \tag{100}$$

hence $\boldsymbol{F_{DFT}} = \frac{1}{N}\boldsymbol{F}$. If we then express the IDFT in terms of the least squares problem, using the cost function defined in equation (93), and the minimisation solution:

$$\left.\frac{\partial \mathcal{J}(\boldsymbol{X})}{\partial \boldsymbol{X}}\right|_{X=X*} = 0 - \boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{y} - \boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{y} + 2\boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{F_{DFT}X} = 2\boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{F_{DFT}X} - 2\boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{y} = 0 \tag{101}$$

$$\boldsymbol{X^*} = (\boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{F_{DFT}})^{-1}\boldsymbol{F_{DFT}^H y} = (\frac{1}{N^2}\boldsymbol{F^H F})^{-1}\frac{1}{N}\boldsymbol{F^H y} = N(\boldsymbol{F^H F})^{-1}\boldsymbol{F^H y} = N\boldsymbol{F^H y} = N\boldsymbol{w^*} \tag{102}$$

Moreover, given that $\boldsymbol{F^H F} = N\boldsymbol{I}$, it follows from equation (100) that $\boldsymbol{F_{DFT}}^{\boldsymbol{H}}\boldsymbol{F_{DFT}} = N\boldsymbol{I} \Rightarrow N\boldsymbol{F_{DFT}}^{\boldsymbol{H}} = \boldsymbol{F_{DFT}}^{-\boldsymbol{1}}\boldsymbol{I} = \boldsymbol{F_{DFT}}^{-\boldsymbol{1}}$. Substituting this into our expression for $\boldsymbol{X^*}$ gives:

$$\boldsymbol{X^*} = N\boldsymbol{I}\boldsymbol{F_{DFT}^H y} = \boldsymbol{F_{DFT}}^{-\boldsymbol{1}}y = N\boldsymbol{w^*} \tag{103}$$

From this we can conclude that least squares solution provided is related to the Discrete Fourier Transform, as our signal y can be estimated using optimum Fourier coefficients as $\boldsymbol{\hat{y}} = \boldsymbol{F_{DFT}X^*} = \frac{1}{N}\boldsymbol{FX^*} = \boldsymbol{Fw^*}$.

b)As previously noted, the Fourier matrix of the Discrete Fourier Transform (DFT) has mutually orthogonal columns, as each column represents a sinusoid at a different frequency. These columns become mutually orthonormal when the additional normalising factor of $\frac{1}{N}$ is included. This is because it holds true that the inner product of any two basis functions of different frequencies are always orthogonal/orthonormal i.e. the inner product is always equal to 0. In the same way as other transforms, the DFT represents a signal as a sum of projections onto its respective basis functions, where in this instance the basis functions are complex harmonically related sinusoids. Hence, application of the Fourier matrix on signal samples changes the basis via a linear unitary transform. Furthermore, given the properties of the Fourier matrix - $\boldsymbol{F^H F} = N\boldsymbol{I}$ - the un-normalised matrix will amplify and rotate the signal in space (rotation being a change of basis). But when normalised - $\boldsymbol{F^H F} = \boldsymbol{I}$ - meaning only a rotation is performed. From the perspective of the least squares problem, the optimisation of Fourier coefficients requires that the weighting of the basis vector components minimise the error in the signal decomposition/representation in the frequency domain, and in turn the time domain after using the inverse DFT.

c) If the input signal to an adaptive CLMS algorithm is seen as a time-domain signal transformed by the Fourier transform, the new input at each time step is a complex phasor. In doing so, the DFT is regarded as a least squares solution and the algorithm named DFT-CLMS. This modified algorithm was applied to the previously used FM signal, and the magnitude of the weights (not the square magnitude as previously) is now used to produce a time-frequency plot. The parameters used were $f_s = 2000Hz$, sample length $N = 1500$ and learning rate $\mu = 1$. As seen on the upper left of figure 42 when there is no implemented 'leak' ($\gamma = 0$), although we can see that the DFT-CLMS algorithm is responsive to the time-variance of frequency but suffers from 'smearing' in the spectrum. The DFT-CLMS algorithm is also known as a 'steady flow DFT', for when $\mu = 1$ and the weights are initialised as 0 the algorithm should be 'stable', and completely forget earlier content after N samples where N is the inputted number of samples presumed to represent one period of the signal. However given that our signal is non-stationary, it has a non-constant time period (or aperiodic) with time-dependent frequency information. Therefore, the DFT-CLMS fails to forget information and does not adaptive sufficiently fast to remove earlier and now irrelevant frequency content from the spectrogram.

In order to combat this, different values of the leakage coefficent γ were tested, which plays the same role as seen in the leaky LMS algorithm previously studied. Introduction of leakage will increase the speed of adaptation by enabling no longer relevant frequency information to be forgotten more quickly, such that it has an inconsequential effect on future DFT-CLMS weights. As seen in figure 42 from the four γ's tested, $\gamma = 0.1$ provided the best balance between forgetting and signal distortion, and outperforms the AR-CLMS approach, particularly at the boundaries between different frequency behaviours e.g. at n = 1000. Too high a leakage coefficient such as $\gamma = 0.5$ results in a widened spectral line defining the relevant frequency content, increasing estimate variance and uncertainty.

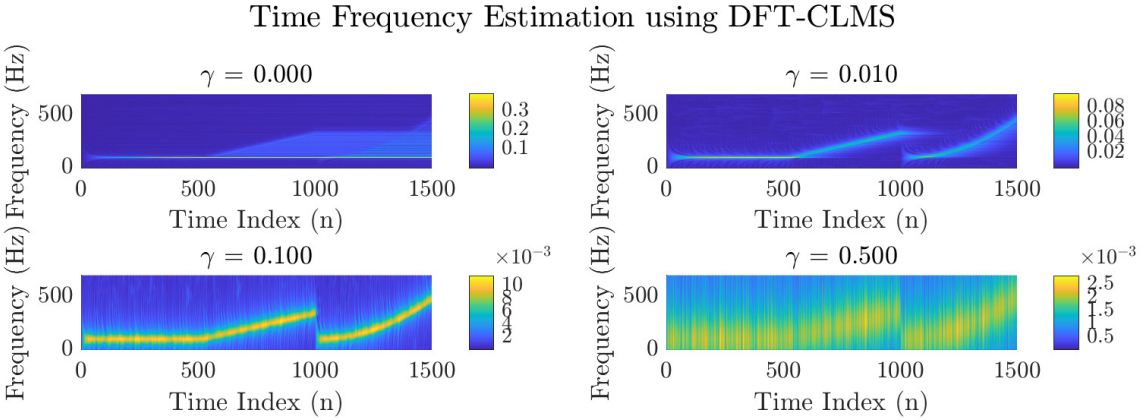


Figure 42: DFT-CLMS with different leakage coefficeints γ performed on a frequency modulated signal, where $\mu = 1$.

d) The DFT-CLMS algorithm was then applied to the first 1200 samples of the EEG POz dataset, using only this segment to minimise computational costs. The time-frequency spectogram is displayed in figure 43, where the learning rate $\mu = 1$ and two leakage coefficients were additionally tested, $\gamma = 0$ and $\gamma = 0.1$. We see that a non-zero leakage coefficient results in distortion and artefacts in the spectrum, such that we are unable to identify the frequency bands present in the EEG signal. In contrast when $\gamma = 0$, the 13Hz SSVEP is clearly present, and the first harmonic at 26Hz can also be distinguished - higher harmonics are not however clear. 50Hz power-line noise is also visible. The non-zero leakge coefficient serves no benefit given the (assumed) stationarity of the 1200-sample EEG segment.

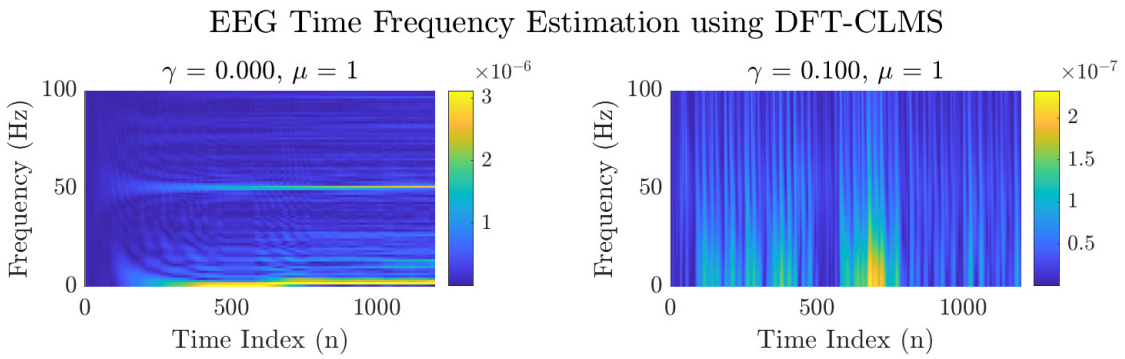


Figure 43: DFT-CLMS performed on EEG data.

4 From LMS to Deep Learning

4.1 Predicting Time Series with LMS

To begin we will continue our study of the behaviour of the LMS algorithm when applying it to non-stationary time-series data. For this a time-series signal $y(n)$ was pre-processed through subtracting the mean and inputted to the LMS algorithm to produce a one-step ahead prediction signal, using a AR model order 4 and learning rate $\mu = 1 \times 10^{-5}$.

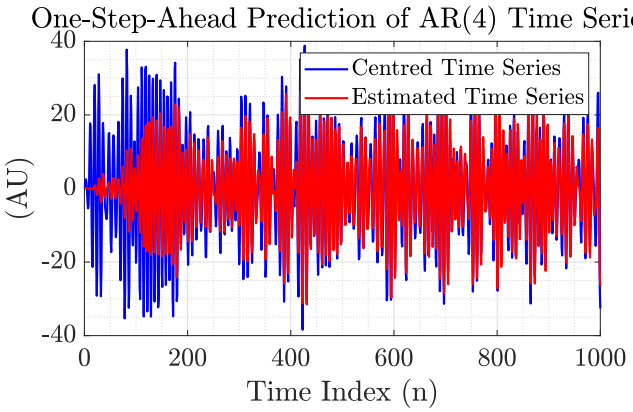


Figure 44: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the entire signal length.

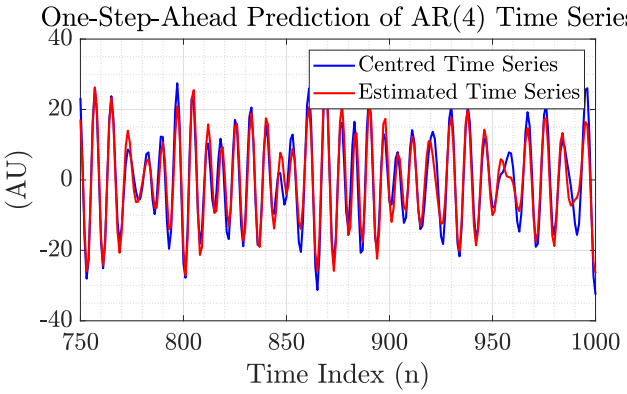


Figure 45: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the last 250 samples.

As seen in figure 44, after a transient period in the first ≈250 samples where the one-step prediction $\hat{y}(n)$ and centred signal $y(n)$ significantly mismatch, for the remaining samples $\hat{y}(n)$ has converged to an improved estimate. As shown in figure 45 the performance of LMS in the final 250 samples is significantly better than the first 250 samples. The MSE between prediction and true signal, for the entire signal length reflected the overall average performance, with value /16.03dB (2.d.p) . The prediction gain $R_p = 10 \log 10 \frac{\hat{\sigma}_p^2}{{e_p}^2}$, defined as the ratio between the variances (powers) of the prediction signal and prediction error corroborates this performance, with a low value of 5.20 dB (2.d.p). In contrast, if only the final 250 samples are isolated the metrics re-calculated, MSE = 11.88dB and $R_p = 9.94$dB, demonstrated the improved prediction capability of LMS over time.

4.2 The Dynamical Perceptron

To capture non-linear features in our signal, a non-linearity known as an activation function is added at the output to improve LMS performance with respect to these non-linear features. In this instance, we use the tanh function, such that:

$$\hat{y}(n) = tanh(\boldsymbol{w(n)}^{\boldsymbol{T}} \boldsymbol{x(n)}) \tag{104}$$

where $\mathbf{w}$ are the weights of the AR(4) model, and $\mathbf{x}$ the 4 immediately past values of $y(n)$, used as input to the LMS algorithm. Consequently, the update rule for the model weights within LMS updates to:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e(n)(1 - tanh^2(\boldsymbol{w(n)^T x(n)}))x(n) \tag{105}$$

where $1 - tanh^2(x)$ was taken from the identity $tanh^2(x) = 1 - tanh^2(x)$. The dynamical perceptron was then used to achieve a one-step ahead prediction of signal $y(n)$.

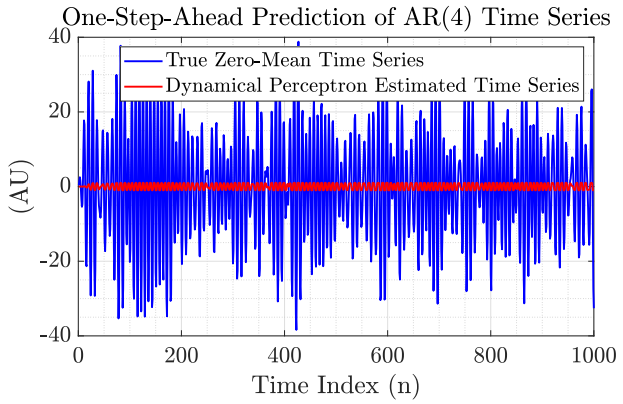


Figure 46: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the entire signal length.

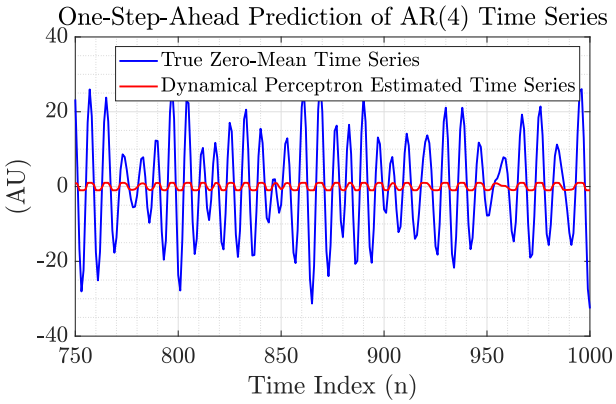


Figure 47: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the last 250 samples.

As seen in figure 46, although the predicted signal is also non-linear and in phase with the maxima and minima of $y(n)$, the estimate poorly models the magnitude. This is because the range of the tanh function is $[-11]$, hence the predicted signal $\hat{y}(n)$ is bounded within these limits, clipping the LMS algorithm output. It also means that values of $y(n)$ beyond this saturate $\hat{y}(n)$, significantly hindering prediction accuracy. The poor prediction capability of the tanh-based dynamical perceptron is quantified with the high $MSE = 22.98$dB (2.d.p) and a large negative $R_p =$ -24.42 dB (2.d.p), where the negative sign implies that the variance (power) of the error signal is greater than that of the prediction signal. Moreover unlike 4.1, isolating the final 250 samples and recalculating the metrics makes little difference, where $MSE =$ 21.96dB (2.d.p) and $R_p =$ -23.27 dB (2.d.p), as the transient period of prediction signals spans much fewer samples ($\approx$ 20-30). Overall, the tanh-dynamical perceptron performs substantially worse that the conventional LMS algorithm used in 4.1; therefore it is not an appropiate model choice in this instance without scaling of the prediction signal or the input signal.

4.3 Scaled Activation Function

A scaling factor α is now included to overcome the range limitations of $tanh \in [-1, 1]$. To mitigate the clipping effects seen in 4.2 α should be set at least to the maximum absolute signal amplitude, which in this instance is : $\alpha \geq 38.8$ (3.s.f). The inclusion of α now results in a prediction value of $\hat{y}(n) = \alpha tanh \boldsymbol{w^T x}$ and the weight update rule:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\alpha \boldsymbol{e}(n)(1 - tanh^2(\boldsymbol{w}(n)^T \boldsymbol{x}(n)))\boldsymbol{x}(n) = \boldsymbol{w}(n) + \lambda \boldsymbol{e}(n)(1 - tanh^2(\boldsymbol{w}(n)^T \boldsymbol{x}(n)))\boldsymbol{x}(n) \tag{106}$$

where $= \mu\alpha$ now becomes the effective learning rate of the LMS algorithm. Consequently, it may be possible that $\alpha \geq 38.8$ could result in system instabilities for larger α values, due to directly increasing the effective learning rate. In fact for $40 \leq \alpha \leq 100$ the MSE and R_p measures were iteratively computed (steps of size 0.1) and plotted in figure 48, to demonstrate that for $\mu = 1 \times 10^{-5}$, increasing α results in monotonic increases in MSE and decreases in R_p. To mitigate this, μ was set to $\mu = 1 \times 10^{-7}$, and the optimal α value was found to equal 92.5 (3.s.f) under the same iterative regime, shown in figure 49.

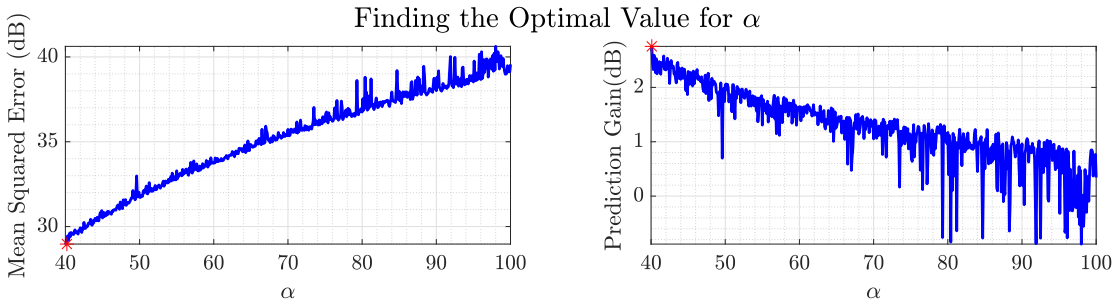


Figure 48: Relationship between MSE and α, and R_p and α for $\mu = 1 \times 10^{-5}$. Red stars indicate the optimal $\alpha \in [40, 100]$. The error monotonically increases to the large effective learning rate .

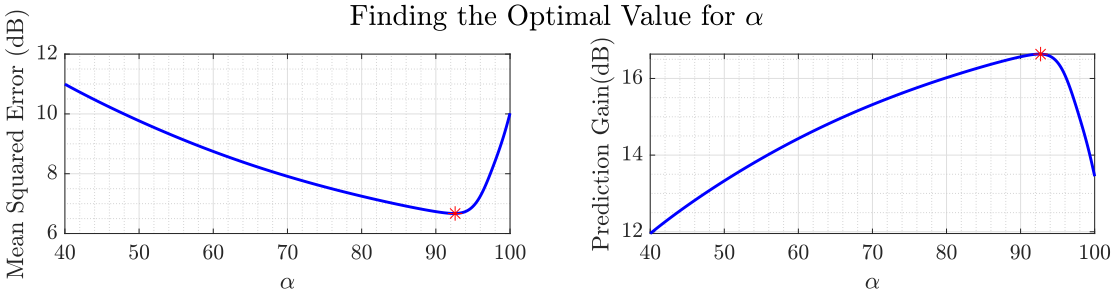


Figure 49: Relationship between MSE and α, and R_p and α for $\mu = 1 \times 10^{-7}$. Red stars indicate the optimal $\alpha \in [40, 100]$. The error is minimised for $\alpha = 92.5$ (1.d.p) with effective learning rate $= 9.25 \times 10^{-6}$ (1.d.p), close to μ in 4.1. and 4.2.

For $\alpha = 92.5$ and $\mu = 1 \times 10^{-7}$, the zero-mean signal $y(n)$ and one-step-ahead prediction $\hat{y}(n)$ are displayed in figure 50.

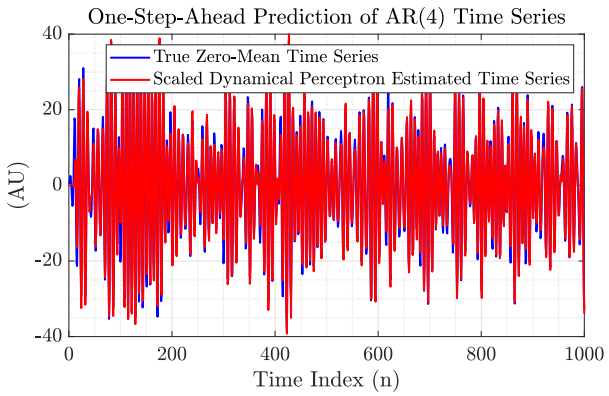


Figure 50: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the entire signal length, when using the scaled dynamical perecptron.

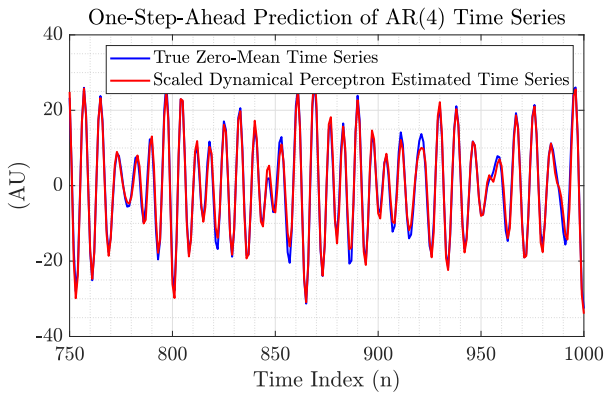


Figure 51: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the last 250 samples, when using the scaled dynamical perceptron.

For the entire signal, $MSE = 6.67$dB (2.d.p) and $R_p = 16.64$ (2.d.p), for which MSE is ~ 2.4 times less than standard LMS in 4.1 and and ~ 3.4 times less than non-scaled tanh in 4.2, on the dB scale. Similarly, R_p is ~ 3.2 times greater than standard LMS. Hence, the performance after scaling the dynamical perceptron is significantly better than previous methods. To avoid having to iteratively find the optimal α, a gradient-descent method for α could be implemented within the LMS algorithm to converge toward an optimum.

4.4 Biased Inputs to Dynamical Perceptrons

As an alternative to centering the signal before inputting to LMS, the input can be biased beforehand, such that the activation function of the dynamical perceptron becomes $\phi(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}+b)$, where b is the bias. This was achieved by augmenting the input such that it becomes 2-by-n array (where n is the number of samples), for which the first row is set to all-ones and the second the signal. Following this, with the LMS algorithm, the weights array is now set to a (order+1)-by-n array, since we now must train a weighting for our bias input, such that the predicted signal becomes appropriately biased by the mean of the true signal $y(n)$. Summarised mathematically, the activation function of the dynamical perceptron becomes, for one step:

$$\hat{y}(n) = \phi\left(\begin{bmatrix} w_0(n) & w_1(n) & w_2(n) & w_3(n) & w_4(n) \end{bmatrix} \begin{bmatrix} 1 \\ x(n-1) \\ x(n-2) \\ x(n-3) \\ x(n-4) \end{bmatrix}\right) \tag{107}$$

$$\hat{y}(n) = \phi(w_0(n) + (w_1(n)x(n-1) + w_2(n)x(n-2) + w_3(n)x(n-3) + w_4(n)x(n-4))) = \phi(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}+b) \tag{108}$$

where b is $w_0(n)$, which must in turn be optimised. Based on the previous iterative regime to optimise α, $\alpha = 75.7$ (1.d.p), while $\mu = 1 \times 10^{-7}$. The resulting one-step ahead prediction and the non-zero-mean signal, based on the biasing method described above is shown in figure 52, and last 250 samples in figure 53.

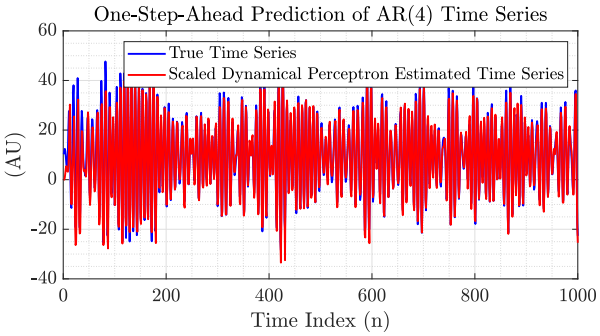


Figure 52: Signal $y(n)$ (blue) and one-step ahead prediction red) for the entire signal length, when using the scaled dynamical perceptron.

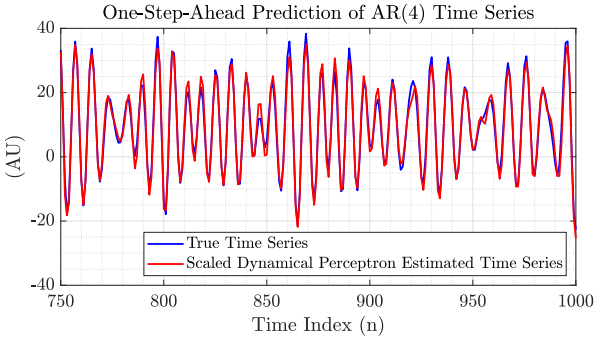


Figure 53: Signal $y(n)$ (blue) and one-step ahead prediction red) for the last 250 samples, when using the scaled dynamical perceptron.

For the entire signal length $MSE = 10.17dB$ (2.d.p) and $R_p = 13.41dB$ (2.d.p), where MSE is 1.6 times less than standard LMS, and R_p is 1.6 times less on the decibel scale. However, the scaled tanh-activation function still outperforms this biased method. This is because the algorithm needs to additionally account for bias, where the model aims to ensure $\hat{y}(n)$ has a mean value equal to $y(n)$. Whilst $mean(y(n)) = 9.9244$, for the one-step-ahead prediction has a lower mean, $mean(\hat{y}(n)) = 9.92582$, showcasing the presence of DC error. Moreover, it is evident from the figures that the model fails to accurately model the largest peaks in $y(n)$.When re-calculating the error metrics for the remaining 250 samples, $MSE = 6.17dB$ (2.d.p) and $R_p = 16.39dB$ (2.d.p) which demonstrates that in the limit, the performance of biased method converges to the performance of the unbiased method in 4.3 where $y(n)$ was centred before input to LMS, as quantified by these metrics. Finally, evolution of the weights of this modified LMS algorithm were plotted to further study convergence. In general, most weights converge by ~ 200 samples, excluding $w0$ representing the bias b, which continues to increase throughout the simulation. The bias weight w0 should converge **in the limit** to a weight that enables the predicted signal to reside closer to the mean of $y(n)$. The remaining weights also converge to lower absolute amplitudes than the unbiased method weights implemented in 4.3, likely explaining the poorer performance of this bias-based model with regards to large amplitudes in $y(n)$.

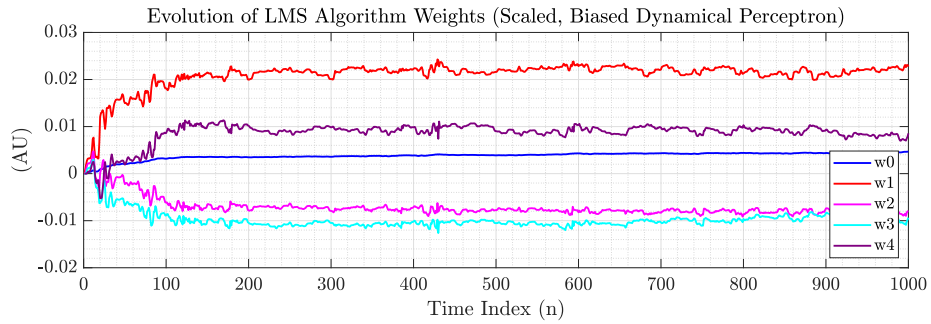


Figure 54: Evolution of filter weights for LMS with an augmented (biased) input.

4.5 Pre-training Model Weights

As mentioned in 4.4 single weight updates on a finite-length signal may not guarantee absolute convergence of our LMS algorithm weights, particularly for the weight w0 corresponding to the bias. To ensure tools such as augmenting inputs with bias are effective, we instead pre-train the weights by overfitting to a smaller number of samples. Pre-training was implemented by beginning with $\boldsymbol{w}(0) = 0$, and iterating over 100 epochs to fit our model to the first 20 samples of $y(n)$ and obtain an initial condition for the latter whole-signal prediction, $\boldsymbol{w_{init}}$. Again, the optimisation regime detailed in 4.3 was used to minimise error, such that $\alpha = 67.9$ (3.d.p) and $\mu = 1 \times 10^{-7}$. The output is shown in figure 55:

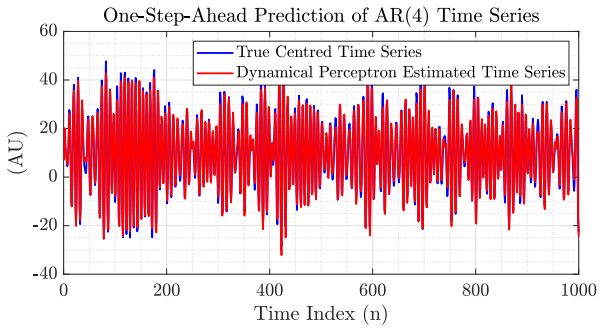


Figure 55: Centred signal $y(n)$ (blue) and one-step ahead prediction red) for the entire signal length, when using pre-training.

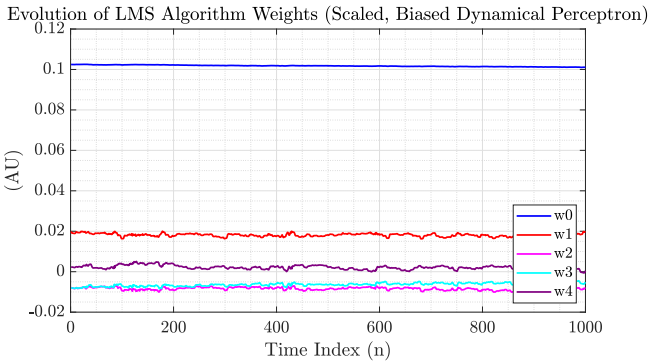
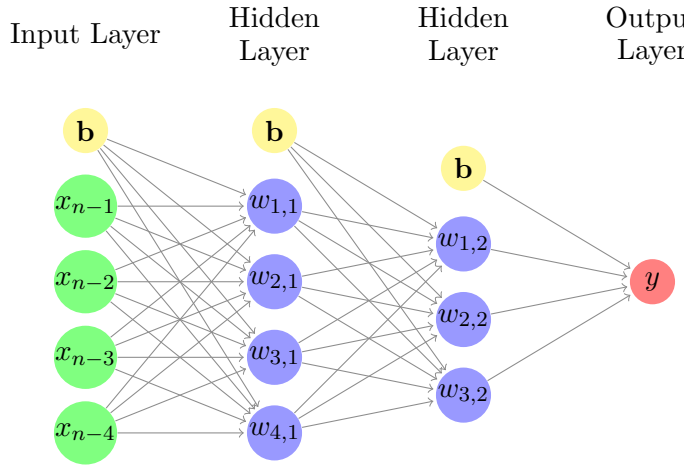


Figure 56: Evolution of filter weights when using pre-training.

Pre-training significantly improves the performance and out-performs all previous methods, with $MSE = 6.41dB$ and $R_p = 16.95dB$, and is therefore has MSE 1.04 times less and R_p 1.02 times more than the metrics of 4.3, the next best output. Furthermore the overfitting procedure increases the speed of convergence where $\hat{y}(n)$ follows $y(n)$ within 5 samples, after a very brief transient overshoot, corroborated by figure 56, where all weights remain close to constant across the 1000 samples. Notably, w0 converges to a much higher value of 0.1 when pre-training is used compared to ~ 0.005 without, making it 20 times larger. The significantly larger value indicates that the bias plays a much greater role in the modelling here; as such 1) the mean of $\hat{y}(n)$, $mean(\hat{y}(n)) = 10.1061$ is closer to that of $y(n)$, $mean(y(n)) = 9.9244$ and 2) the algorithm is more accurate when modelling the largest amplitudes of $y(n)$.

4.6 The Backpropagation Algorithm

Below we consider a 2-hidden-layer deep neural network. These networks consist of: 1)Neurons - nodes with activation values 2) Connections between neurons - weights 3) Biases - associated with each neuron.



To progress inputted data from the input layer to the output layer, a **'forward pass'** is performed. A 'forward pass' involves iterative use of an update formula, in which we compute the values associated with neurons, known as **activations**. The general formula involves computing the activations of neurons in layer l based on the weighted activations of neurons in layer $l-1$, which are in turn typically subjected to a highly non-linear activation function, such as $tanh$, as shown below:

$$\begin{bmatrix} {a_0}^1 \\ {a_1}^1 \\ \vdots \\ {a_n}^1 \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{00} & w_{01} & \dots & w_{0k} \\ w_{10} & w_{11} & \dots & w_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j0} & w_{j1} & \dots & w_{jk} \end{bmatrix} \begin{bmatrix} {a_0}^0 \\ {a_1}^0 \\ \vdots \\ {a_n}^0 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right) \tag{109}$$

or in matrix form:

$$\boldsymbol{a}^{(l)} = \sigma(\boldsymbol{W}\boldsymbol{a}^{(\boldsymbol{l-1})} + \boldsymbol{b}) = \sigma(\boldsymbol{Z}) \tag{110}$$

where $\boldsymbol{a}$ denotes the activations,$\boldsymbol{W}$ the weights and $\boldsymbol{b}$ the biases. Repeated use of this equation enables us to advance toward the output layer. Upon arrival at the output layer the output, which is our prediction of the signal $\hat{x} = \boldsymbol{a}^{\boldsymbol{L}}$ is evaluated by a cost function (J) to assess the error of the network. This for instance could be the Mean Squared Error (MSE) measure. However, once this error is calculated, the network should use this error to be able to learn and hence improve it's performance. Achieiving this requires the implementation of **backpropagation**, which is used to iteratively update the weights $\boldsymbol{W}$ and bias $\boldsymbol{b}$ of our network in order to tune the activations associated with neurons, 'propagating' backwards while doing so from the output layer back to the input layer. Once the propagation has finished, new data is inputted, activation values computed, the cost function determined and the weights/bias again updated. These forward and backward passes persist until all training data is used, or convergence of W to a presumed global optimum is reached.

More specifically, the role of the backpropagation algorithm in achieving this is to quantify what is the change in J relative to the network parameters w,and b? By doing this, we can deduce how these parameters should be changed in order to optimise the cost function. The 'change' is computed through partial differentiation, more formally called a **'gradient descent'** procedure. However, since the cost function is a function only of the activation value at the final layer a^L (or otherwise the prediction), the chain rule is applied. Furthermore, as we propagate back through the network, the partial differentials of earlier layers will have dependencies on the calculations of later layers. For instance, for the depicted neural network with two hidden layers, to calculate the rate of change of the cost function with respect to the weights of the first hidden layer:

$$\frac{\partial J}{\partial w^1} = \frac{\partial J}{\partial a^3}\frac{\partial a^3}{\partial z^3}\frac{\partial z^3}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial a^1}\frac{\partial a^1}{\partial z^1}\frac{\partial z^1}{\partial w^1} \tag{111}$$

where as mentioned, the chain rule is exploited to give the chain of dependencies required find a relation between w_1 and J. This means that error information from the latter end of the network propagates back to the start, hence the coined name 'backpropagation algorithm'. A similar partial derivative for the cost

function with respect to bias values b. Upon computing these partial differentials, the weights and bias of the neural networks are then updated in the opposite direction to the gradient, weighted by learning rate α:$w^l = w^l - \alpha\frac{\partial J}{\partial w^l}$ (and same form for bias). Consequently, through computation of the gradients i.e. the partial derivatives and the use of gradient descent in the backpropagation algorithm, the cost function J can be iteratively minimised from one epoch to the next, which permits the training of a deep neural network.

4.7 Deep Neural Networks

To evaluate the capabilities of deep neural networks (DNNs) in accurately modelling highly non-linear signals, we use a non-linear signal composed of 10 separate non-linear sinusoidal components, with varying amplitude, frequency and phase relative to one another. The overall signal $\boldsymbol{x}(n)$ is then subjected to an unknown activation function such that $y(n) = \phi\{\boldsymbol{x}(n)\}$. Both $\boldsymbol{x}(n)$ and $y(n)$ are displayed in figure 57.

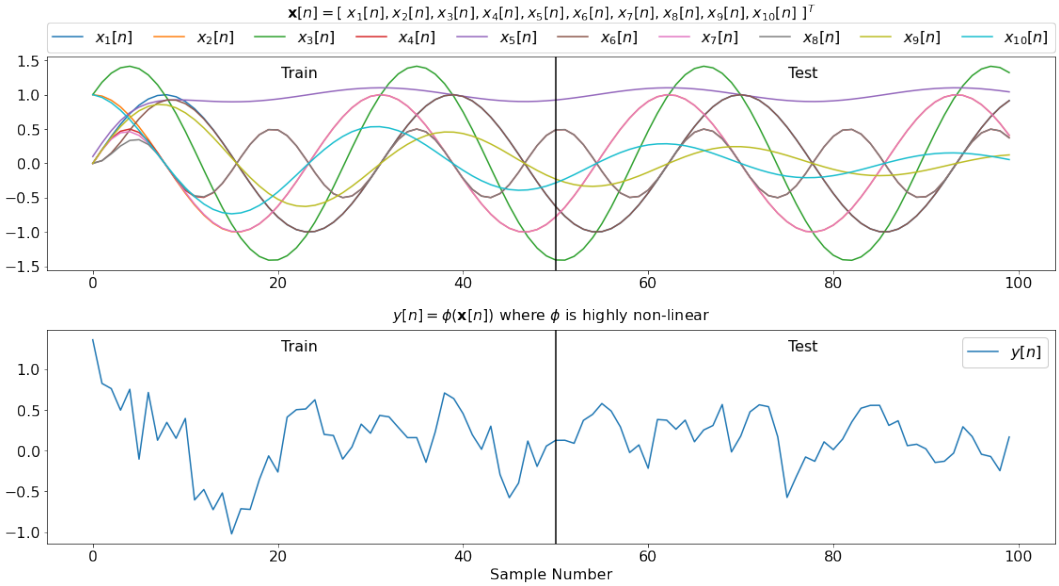


Figure 57: Top: All components of input signal $\boldsymbol{x}(n)$. **Down**: $y(n)$ as generated by a highly linear unknown activation function.

Primarily in this question, we investigate the performance of 3 models in approximating $y(n)$: 1) Linear Dynamical Perceptron 2) Tanh-based Dynamical Perceptron (as previously investigated) and 3) 4-hidden-layer DNN with the Rectrified Linear Unit function (ReLu). ReLU is a piecewise linear (hence globally non-linear) function that is used primiarly for reduced computational costs, and it's robustness to the 'vanishing gradient' phenomenon that plagues other network models. Across 2000 epochs, with learning rate $\mu = 0.01$ and noise power of 0.05, the model performances were trained and evaluated on a testing set, quantified with the Mean Squared Error. The prediction signals from these methods are shown in figure 58 and their respective MSE curves in figure 59. In all cases, and referring to table 2, we see that the training loss (MSE) decreases and plateaus reaching the minimum value at the final epochs. In contrast, the testing loss reaches a minimum value in all cases much earlier, and earliest for the deep ReLU-based neural network, after only 496 epochs which is ~ 4.8 times and $\sim$ faster than for the simple (single neuron) and dynamical (tanh) perceptron models respectively. Furthermore, the minimum MSE is lowest for the deep neural network and highest for the simple perceptron model, due to greater complexity/degrees of freedom of the neural network enabling highly non-linear transformation of input to output.

The minimum error index here is a valuable metric given that training may cease upon reaching this point (although requires prior knowledge). However, MSE increases beyond these minimum indices, which could be explained by the constituent of MSE where $MSE = bias^2 + variance$. Before the minimum index, it is likely that the bias is decreasing faster than the variance (underfitting), resulting in a net decrease in

MSE. But beyond the minima, the bias plateaus and variance continues to increase as a result of model overfitting. Hence, a balance must be struck between bias and variance to control this error. Notably, there is a sudden increase in MSE after acheiving the MSE minimum for the ReLU unit, most likely since it is most prone to overfitting effects given the many additional degrees of freedom afforded to a deep neural network. This also causes a lower initial convergence speed in the training phase compared to the simple and dynamical perceptron models.

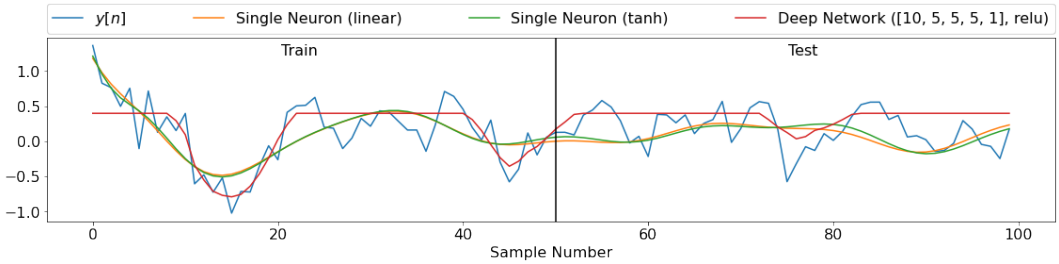


Figure 58: True signal $y(n)$ and the predictions of the 1) Simple Perceptron Model (orange), 2) Tanh Perceptron Model (green) and 3) ReLU DNN Model (red)

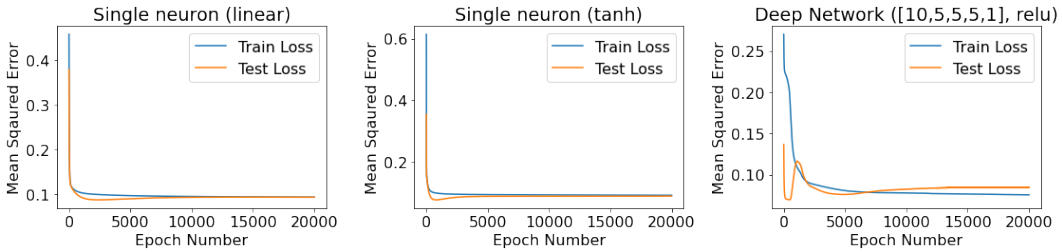


Figure 59: True signal $y(n)$ and the predictions of the 1) Simple Perceptron Model (orange), 2) Tanh Perceptron Model (green) and 3) ReLU DNN Model (red)

Model	Min Training MSE	Min Training MSE Index	Min Testing MSE	Min Testing Index
1	0.094	20000	0.088	2402
2	0.092	20000	0.077	803
3	0.075	19996	0.069	496

Table 2: Minimum MSE and corresponding index in training and testing data, for tested models, where 1) Simple Dynamical Perceptron 2) Tanh Dynamical Perceptron and 3) ReLU DNN

4.8 Effects of Noise Power on Deep Learning

The above analysis is repeat for two different noise powers of 0.005 and 0.5 (10 times lower and higher than previously). For noise power of 0.005, overall the models experience less overfitting. This is evident from figre 60 and table 3 given that the minimum MSE values in testing are achieved at epochs much later in the series, and smaller incresaes in MSE beyond this are seen. Consequently, the variance increases less than previously after reaching the minimum, and hence less overfitting. This can also be seen when observing the prediction signals, that the three models do not follow peaks and troughs in noise, but more generalised amplitude content. The deep neural network has the lowest minimum MSE in testing of 0.024, ~ 3.8 times lower than that of the simple perceptron model, again due to the additional degrees of freedom it possesses permitting non-linear input transformation. It does also again transient increase in MSE after 2500 epochs, which could be associated with the bias-variance trade off. An additional contributing factor could be the backpropagation algorithm itself, wherein early fluctuations in network weights produce a transient increase in MSE.

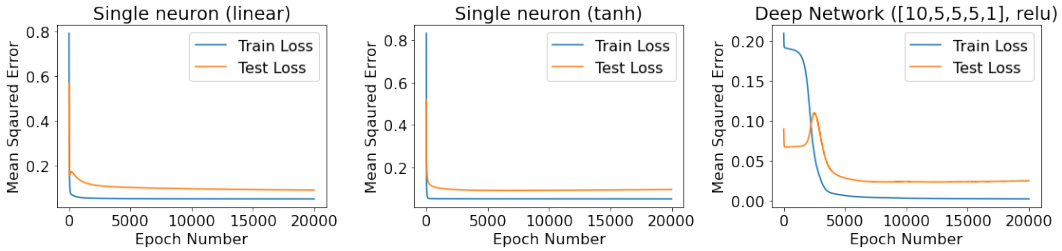


Figure 60: **Noise Power = 0.005**: True signal $y(n)$ and the predictions of the 1) Simple Perceptron Model (orange), 2) Tanh Perceptron Model (green) and 3) ReLU DNN Model (red)

Model	Min Training MSE	Min Training MSE Index	Min Testing MSE	Min Testing Index
1	0.053	19999	0.091	6757
2	0.037	19999	0.092	19999
3	0.052	19999	0.024	12048

Table 3: **Noise Power = 0.005**: MSE data for evaluated models.

When noise power is increased to 0.5, all models see a significant decrease in performance, shown in figure 61, (with MSE values up to 20 times higher than power of 0.005) , where the performance is now worst for the deep neural network. This is because the additional degrees of freedom of the network majorly overfit to the noise in the inputted training data such that there is a large increase in variance in the weights of the network as it tries to adapt to new noise information in the testing phase. Overfitting also significantly affects both the single and dynamical perceptron, as the minimum MSEs are achieved at epochs and 16 and 284 respectively meaning little valuable learning is performed before the effects of variance override the models, as per table 4. In this scenario, the linear, simple perceptron model infact has the lowest end-error of 0.646 and is more robust to noise given it's inability to produce non-linear mappings of noise behaviour from input to output. Overall, deep learning is advantageous for the prediction of highly non-linear data, but dependent on the training data quality. With too great a noise power, deep neural networks suffer from overfitting and hence become unable to generalise to testing datasets. However, there are methods to increase robustness, such as 1) regularisation 2) increasing the training set size (to allow the network to better learn the underlying phenomenon for data behaviour) or 3) cross-validation.

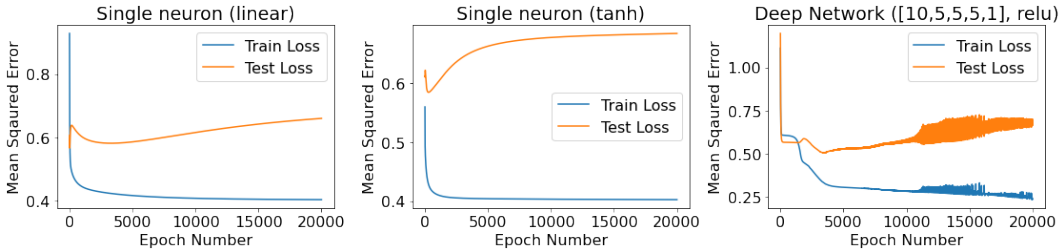


Figure 61: **Noise Power = 0.5**: True signal $y(n)$ and the predictions of the 1) Simple Perceptron Model (orange), 2) Tanh Perceptron Model (green) and 3) ReLU DNN Model (red)

Model	Min Training MSE	Min Training MSE Index	Min Testing MSE	Min Testing Index
1	0.404	19999	0.585	16
2	0.566	19998	0.231	284
3	0.403	19979	0.504	2453

Table 4: **Noise Power = 0.5**: MSE data for evaluated models