Adaptive Signal Processing and Machine Intelligence

Coursework

ELEC70001

(Trevi Yek) - CID02145518

Spring Term Coursework

IMPERIAL

March 31, 2025

Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

1.1.1 Approximation in the definition of PSD.

The 2 definitions of PSD are:

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

and

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

Starting from eqn2, the squared, absolute inner sum is rewritten using the the multiplication of a complex and conjugate term.

$$P(\omega) = \lim_{N\to\infty} E\{\frac{1}{N} \sum_{n=0}^{N-1} x(n)e^{-jwn} \sum_{m=0}^{N-1} x^*(m)e^{jwm}\} \tag{3}$$

Factorization

$$P(\omega) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} E\{x(n)x^*(m)\}e^{-jwn}e^{jwm} \tag{4}$$

The expectation term is substituted with autocovariance.

$$P(\omega) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} r(n-m)e^{-jw(n-m)} \tag{5}$$

For further simplification, a dummy variable $k = n - m$ is used. Following this, the lower and upper bounds of the summation operator are adjusted in terms of k. The number of unique combinations of k is also considered as a multiplication factor.

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

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

The second term is the covariance sequence that is under a mild assumption of rapid decay.

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k|r(k)e^{-jw(k)} = 0 \tag{8}$$

Hence the PSD can be approximated to the definition provided in eqn1.

$$P(\omega) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} Nr(k)e^{-jw(k)} = \lim_{N\to\infty} \sum_{k=-(N-1)}^{N-1} r(k)e^{-jw(k)} = \sum_{k=-\infty}^{\infty} r(k)e^{-jw(k)} \tag{9}$$

To simulate comparisons between both definitions, an impulse signal and a sinusoidal signal was used to represent rapid decay and slow decay in the biased ACF respectively.

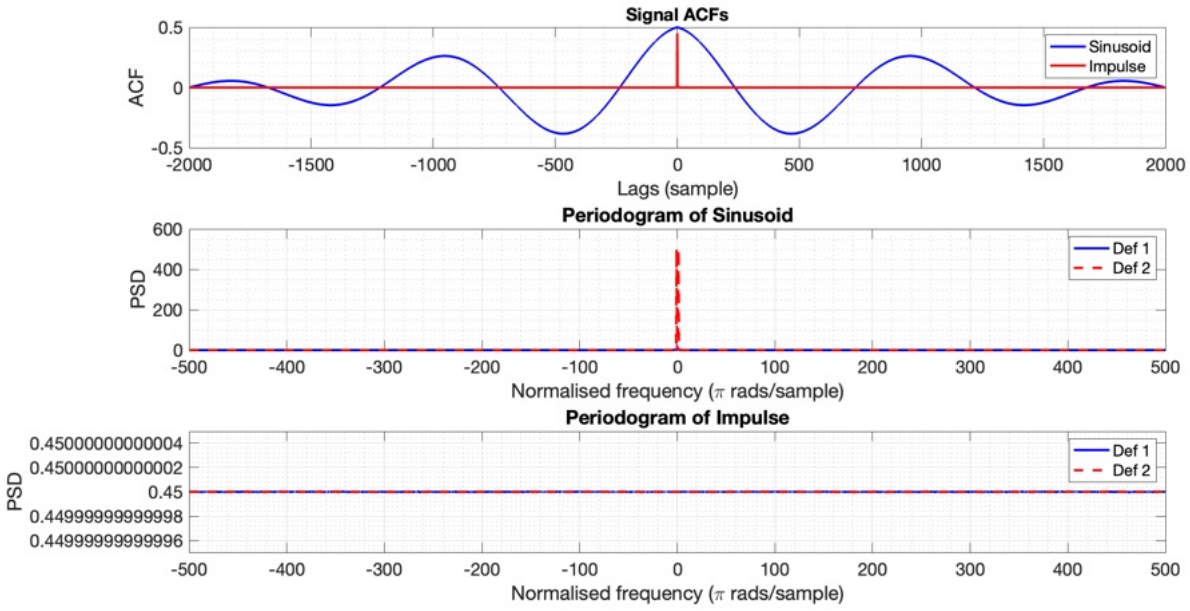


Figure 1.1: ACF and PSD estimates for both definitions

In the case of the impulse signal, the ACF decays rapidly so $Def1 = Def2$. The ACF of the sinusoid decays slowly, so $Def1 \neq Def2$ around the normalised frequency of 0.

1.2 Periodogram-based Methods Applied to Real–World Data

1.2.1 The perception of the periodicities in the data

Starting from the raw sunspot data, subtracting the mean gives 0 power at 0Hz. Detrend removes the best-fit line from the dataset. A very similar mean removed and detrended data implies that the mean of the signal does not change over time (stationary). The use of log and mean removal allows for more useful peak information. The periodogram was determined through the application of both a rectangular window and Hamming window.

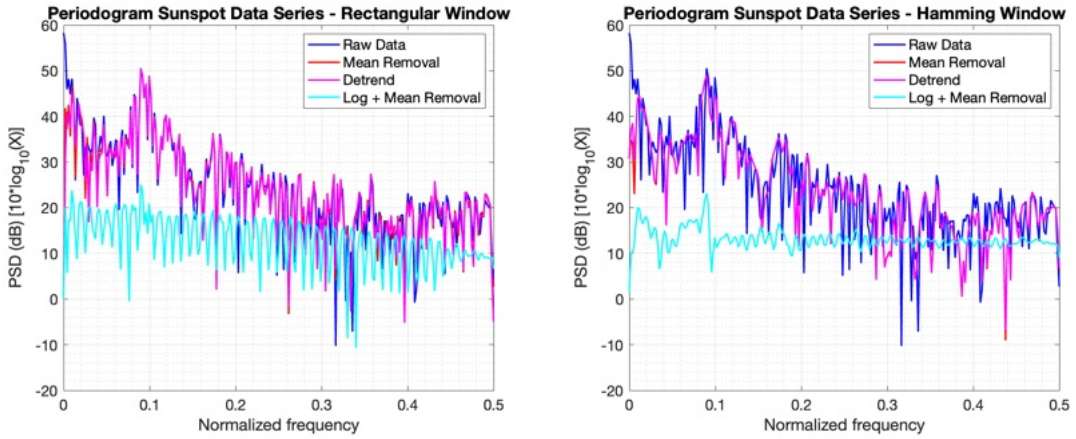


Figure 1.2: Periodogram using Rectwin and Hamming

Detrend removes the best straight-fit line from the sunspot dataset. So 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, implying non-stationary. However, in this instance, we see that both are highly similar. The application of a logarithm to the data, followed by mean subtraction produces significantly different results to the prior two processing methods. Note that, to avoid logarithms of zero

values, the periodogram values < 0.001 is replaced with 1. The logarithm compacts the data into a smaller scale, smoothing the PSD and allowing useful peak information to be seen more clearly.

Standard and Bartlett methods are used to determine the periodogram of the provided EEG signal. Looking at Figure 1.3, the application of Bartlett's method via "pwelch", Welch's method implementation in MATLAB, reveals several distinct peaks: a wide peak at 8-10Hz corresponding to subject fatigue, and narrow peaks at 50Hz, 13Hz, 26Hz and 39Hz, which are the DC noise, and the fundamental frequency of the SSEVP (f_0^{SSEVP}), and it's harmonics (f_i^{SSEVP}) in the same order. The 3rd harmonic at 52Hz is difficult to identify for all windowing sizes due to the close proximity of the high-amplitude DC $50Hz$ mains component.

1.2.2 The basis for brain computer interface (BCI)

When we compare the standard periodogram to 10s window length in Bartlett's method, variance in the PSD is observed to reduce, however bias is introduced. This gives us a trade-off between bias and variance. This effect is more clearly observed in between the standard and 1-s windowing periodograms (Figure 1.4). A smaller window length provides more segments for calculation, reducing variance. This is evident in the increased smoothness and increase bias as window length decrease.

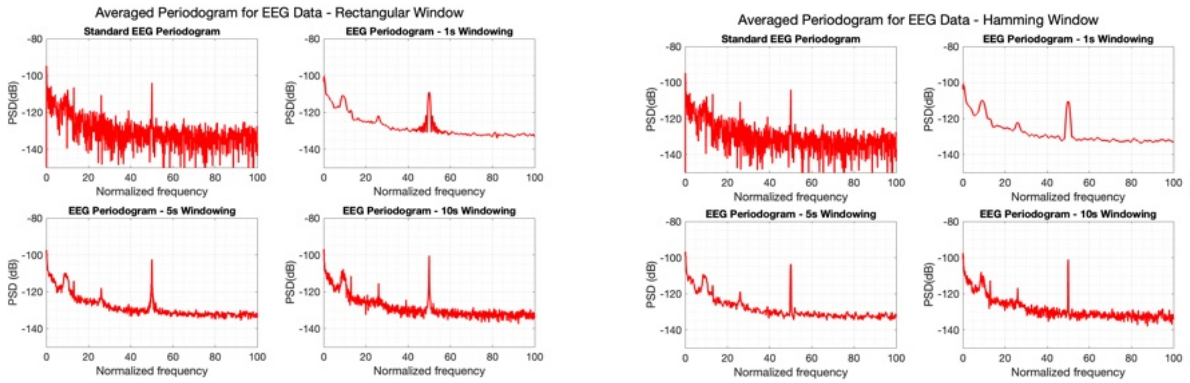


Figure 1.3: EEG Periodogram with Rectwin and Hamming

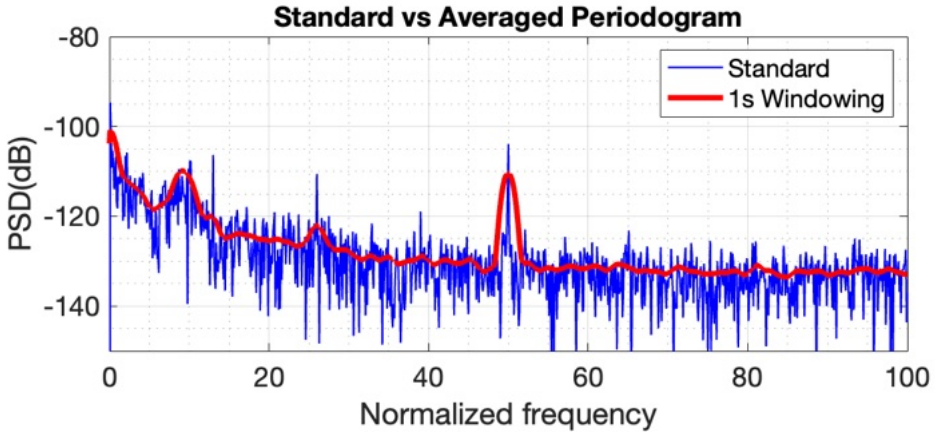


Figure 1.4: Standard and averaged peiodogram comparison

1.3 Correlation Estimation

1.3.1 Unbiased correlation estimation and preservation of non-negative spectra

The *correlogram spectral estimator* is calculated using the unbiased and biased ACF estimates of a signal. For a white Gaussian noise(WGN), a noisy sinusoidal signal and filtered WGN, the biased and

unbiased estimates are expressed by:

$$\textbf{Biased: } \hat{r}_k = \frac{1}{N} \sum_{n=k+1}^{N} x(n)x^*(n-k) \tag{10}$$

$$\textbf{Unbiased: } \hat{r}_k = \frac{1}{N-k} \sum_{n=k+1}^{N} x(n)x^*(n-k) \qquad (0 \leq k \leq N-1) \tag{11}$$

In Figure 1.5, for both biased and unbiased ACF estimates when $|k| < 500$, the estimates are very similar. However, beyond this value of k, there is a stark difference between the plots where the biased ACF estimate decays to zero and the unbiased ACF estimate has an increase in amplitude. We should also note that the biased estimate provides a more accurate representation because the expected ACF estimates are: flat spectrum for WGN, delta train with impulses corresponding to sinusoidal frequencies for noisy sinusoid, flat spectrum with lower amplitude at higher frequencies for filtered WGN (compared to the original WGN). The unbiased estimate contains large negative amplitude components for all the signals. These negative values come from how the unbiased estimate is computed by convolving the true correlogram/PSD with a rectangular window filter, whilst the biased estimate is based on convolution with a triangular Bartlett window.

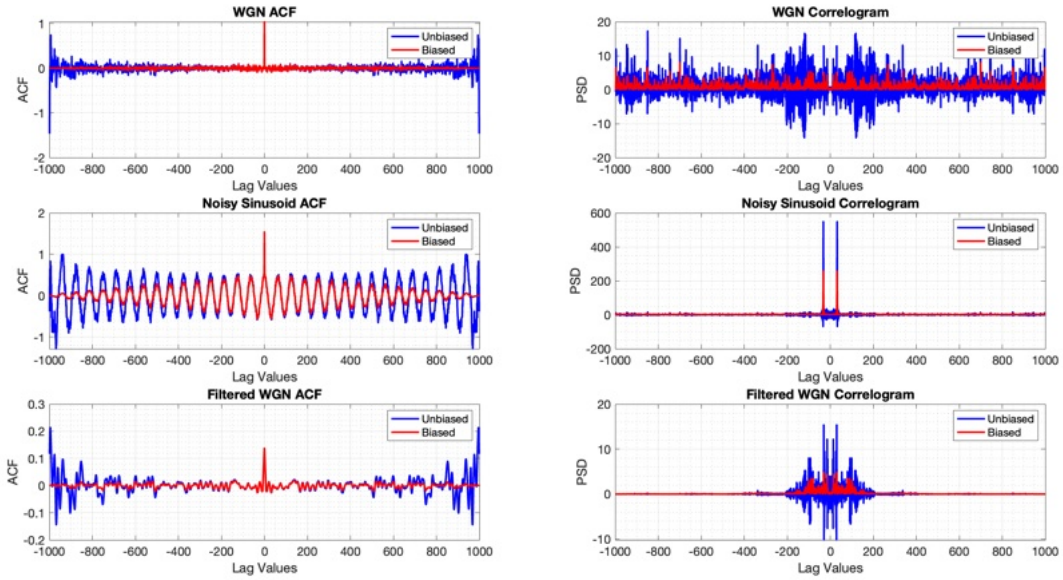


Figure 1.5: ACF and Correlograms (PSD) for WGN, noisy sinusoid and filtered WGN

Using my code from the previous section, 100 realisations of a random process was simulated as described by a combination of three sinusoids, in addition to WGN:

$$x(n) = 1.5\sin(2\pi n) + 0.5\sin(0.3 \times 2\pi n) + 1.2\sin(1.8 \times 2\pi n) + \eta(n) \tag{12}$$

Hence three peaks should be observed in the PSD spectrum at the frequencies 1Hz, 0.3Hz and 1.8Hz, as shown in Figure 1.6. Some rippling is observed around the peaks due to the nature of the biased ACF, previously mentioned to correspond to a triangular Bartlett window, where the its frequency (Fourier) domain representation is equivalent to a squared, periodic sinc function. The amplitude of the spectra further from the peaks converges to 1, which is the true amplitude of the PSD of WGN, demonstrating how the biased calculation of PSD is asymptotically unbiased. The standard deviation takes a similar shape to the spectrum itself, where significant variance is seen at the peaks. So it is an inconsistent estimator.

1.3.2 Plotting the PSD in dB

The same realisations are plotted on a decibel scale (Figure 1.7). The log scale compresses the range of values and reveals any exponential relationships. We can see that the spectral peaks are attenuated and slightly accentuated the noise peaks. This reduces the variance around the spectral peaks and keeps the variance of noise peaks high, making it easier to identify the peaks of interest. Hence the dB representation of PSD is important if there is a low-amplitude peak that would be difficult to distinguish from noise in the normal scale.

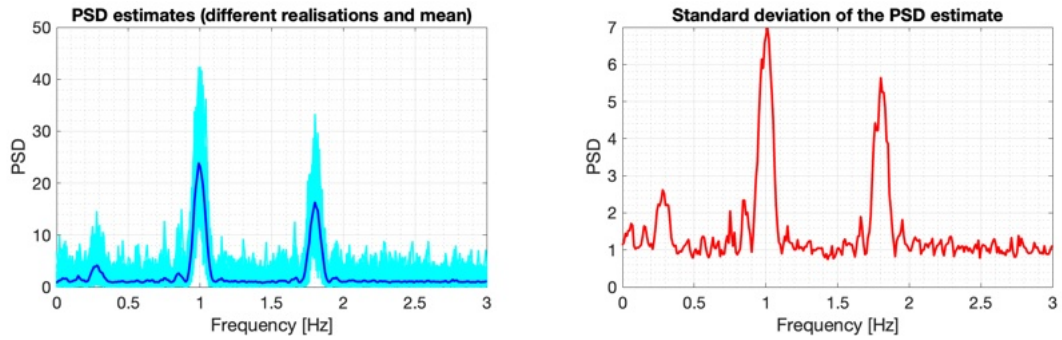


Figure 1.6: Plotting the PSD for noise sinusoids composition

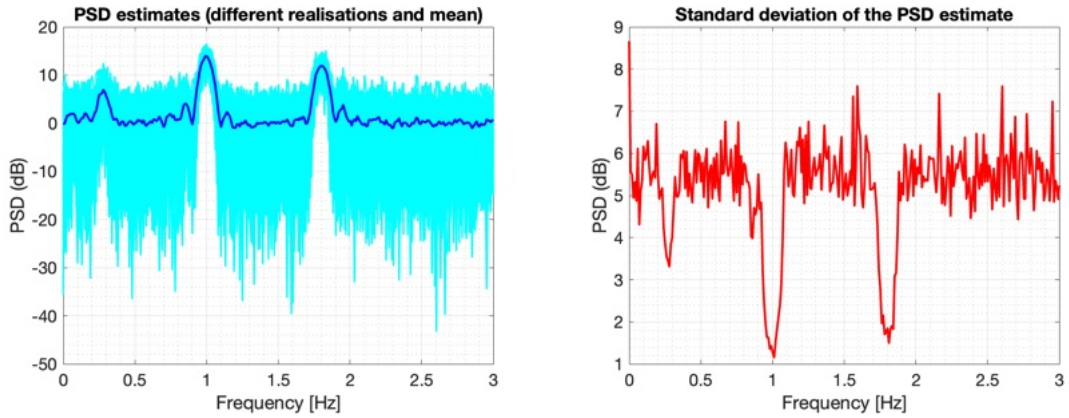


Figure 1.7: Plotting the PSD for noise sinusoids composition in dB

The complex exponential signal was simulated for varying numbers of samples, varying the frequency resolution. The periodogram distinguishes the two spectral peaks, only when the resolution of the frequency axis is larger than the difference in frequency peaks. Increasing the number of samples converges the periodogram into the true line spectra, where there are two frequency peaks (Figure 1.8).

Again, noting that the periodogram is derived from a biased estimate, i.e. with a Bartlett window, it introduces a smoothing effect, which limits the ability to distinguish narrowband frequency components, that are present in our complex exponential signal. For such window, the resolution is approximately $0.89/N$, so the minimum sample size for the given frequencies is:

$$\Delta f = \frac{0.89}{N} \quad N = \frac{0.89}{\Delta f} = 44.5 \approx 45$$

Figure 1.8 shows this result where differentiation of frequency peaks happen from around $N = 45$.

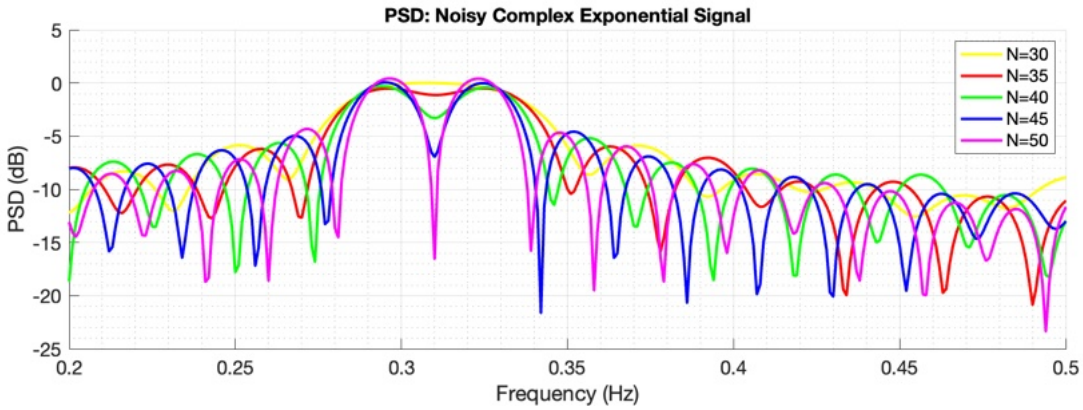


Figure 1.8: Periodogram of complex exponential signal for varying sample lengths

1.3.3 Frequency estimation by MUSIC

The Multiple Signal Classification Method (MUSIC) can be used to find the line spectra of our simulated exponential signal, which is classified as a subspace method, in which eigendecomposition is performed on the autocorrelation matrix of time series data.

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

Line 1: 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, and the shape of matrix X used to determine the autocorrelation estimate R where it is equivalent to the matrix multiplication of the pseudo inverse of X, by X ($R = X^T X$).The input 'mod' defines that the matrix X will be deduced from forward and backward error estimates.
Line 2: Is the in-built implementation of MUSIC that returns S, the psuedospectrum of the input vector x, and the frequency vector F. The output square autocorrelation matrix R from the first line is provided as input to pmusic, which determines the eigenvalues of R and sorts them in descending order. $p = 2$ is the signal subspace dimension or the number of complex exponentials that build the simulated signal vector x. The MUSIC algorithm assumes that the vector input x in the form of the autocorrelation matrix R, is composed of complex exponentials (i.e. uncorrelated sources) as well as WGN.

$$\mathbf{R} = \mathbf{A}\mathbf{R}_\mathbf{s}\mathbf{A}^H + \sigma^2\mathbf{I} \tag{13}$$

where $\mathbf{A}$ is the mxp matrix containing the p subspace vectors $[e_i, ..., e_p]$ of the full signal vector $[1, e^{2j\omega}, ..., e^{(m-1)j\omega}]$. H is the Hermitian transpose and $\sigma^2\mathbf{I} = \mathbf{R}_n$ (noise). The subspaces are orthogonal hence the algorithm can be expressed by.

$$\hat{P}_{music}(\omega) = \frac{1}{\sum_{i=p+1}^{m} |e^H v_i|} \tag{14}$$

where $v_{(i,...,p)}$ are the noise eigen vectors. The inner product between the source signal and noise eigenvectors will be zero, hence the estimate will have p spectral peaks. Finally, [] sets the FFT default length of 256, 1 is the sampling frequency, and 'corr' gives context that the input matrix is an autocorrelation matrix estimate.
Lines 3 and 4: 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 $0.25 - 0.40Hz$.

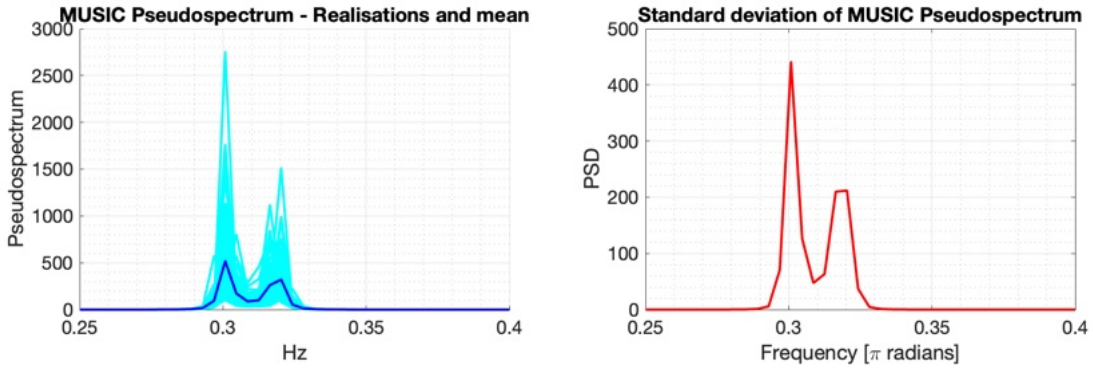


Figure 1.9: Pseudospectrum of simulated complex exponential using the MUSIC

Figure 1.9 is the result from the MUSIC algorithm for sample length $N = 30$ and the two spectral peaks can be resolved although the standard periodogram is incapable to do so. However the disadvantages include the fact that it relies on known model order, which is usually unlikely and failure to do so causes great deterioration in accuracy, and the variance in the peaks are significantly higher. So to use MUSIC, preprocessing work has to be done to determine mode order such as using AIC and MDL.

1.4 Spectrum of Autoregressive Processes

Popularly, the autoregressive (AR) process parameters may be determined using the vector-matrix form of the Yule-Walker equations, where $r_{xx} = R_{xx}a$ and R_{xx} is the ACF matrix of the signal in question. In order to do this R_{xx} must be invertible and positive definite. Hence comparing biased and unbiased ACF estimates, they are similar in small lags and with larger lags 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 the matrix becomes non-invertible.

The ARMA process below was generated the results are plotted in Figure 1.10 and 1.11.

$$x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n) \tag{15}$$

where w represents Gaussian noise with mean 0 and variance 1.
The sample length was first set to 1000 and then later 10,000, for which the PSD's are provided in Figures 1.10 and 1.11 (where the first 500 samples were removed due to transient filter effects). As expected AR(2) is unable to effectively model the true PSD for both cases. For $N = 1000$, AR(4) is inadequate for modelling the twin peaks, while AR(4) in $N = 10,000$ can be used to do so, suggesting the importance of adequate resolution in the power spectra on top of the model order. The higher order model orders are sufficient to model the two peaks, although the tails are not fitted as well. Increasing the order from these values does not offer improved performance in estimation.

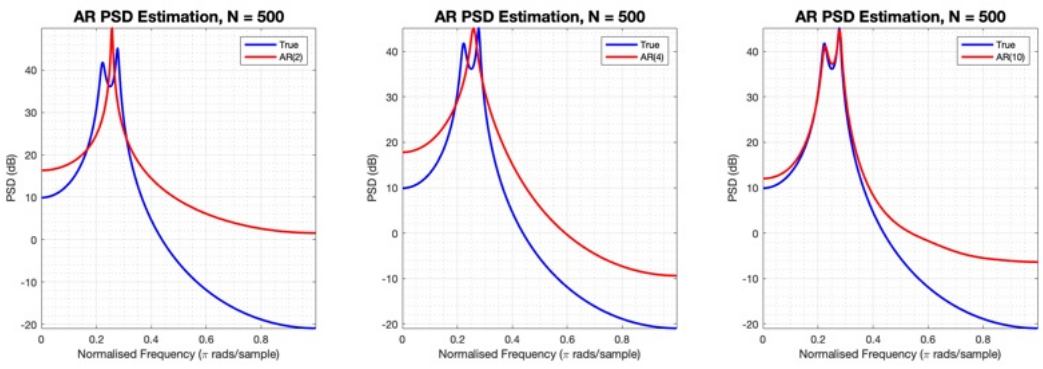


Figure 1.10: Modelling of a AR process using 1000 samples

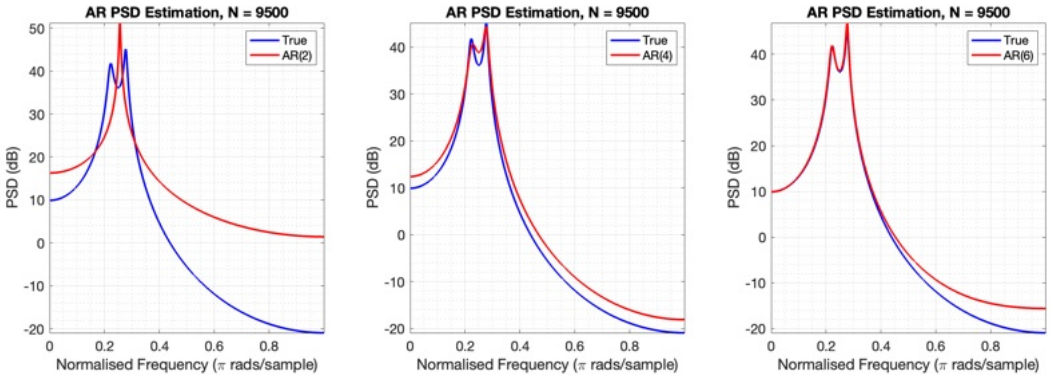


Figure 1.11: Modelling of a AR process using 10,000 samples

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

The standard and averaged periodograms for the RR intervals, which are the time interval between two R peaks of ECG QRS complex, was plotted calculated and presented in Figure 1.12. The rectangular windows were used for lengths 150s and 50s, usng the *pwelch* function.

Differences between the RRIs: The maximum peaks in the RRIs are $0.172Hz$ (10.32bpm) for RRI1, $0.418Hz$ (25bpm) for RRI2, and $0.125Hz$ (7.5bpm) for RRI3. Note that RRI3 displays several significant harmonic peaks (approximately $0.25Hz$ and $0.375Hz$).
Differences caused by window length: Since windows are set through *pwelch* (equivalent to using the Bartlett's method). As expected, smaller window lengths decrease variance but increase bias for all three trial data.

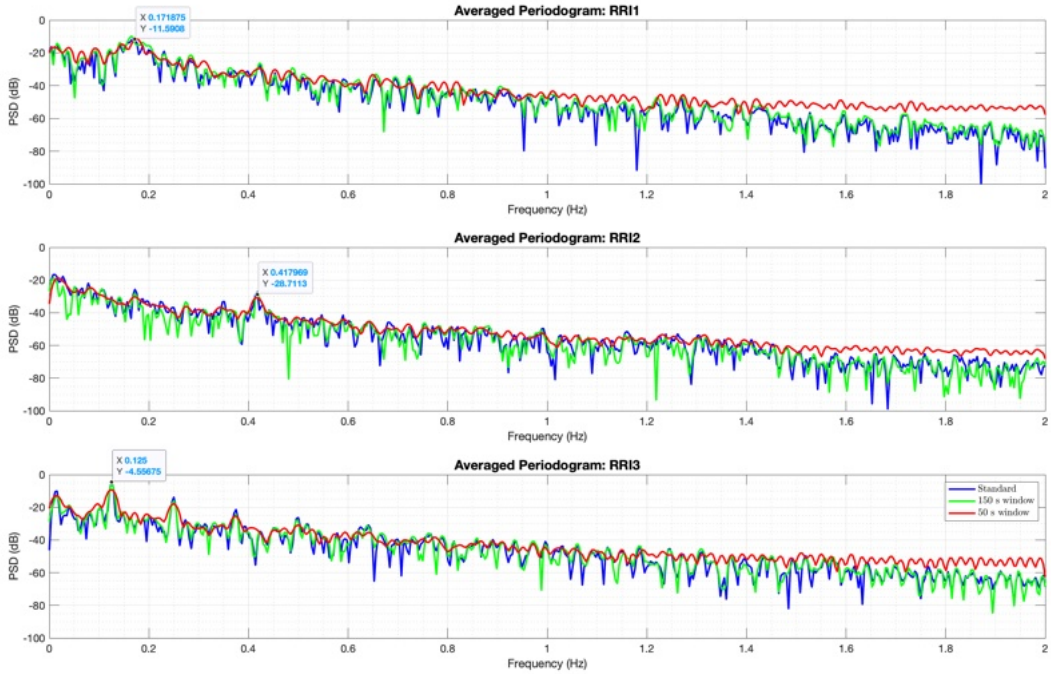


Figure 1.12: PSD for standard and averaged periodogram with different window lengths

The AR spectrum estimates are computed and displayed in 1.13. The model orders are increased until the first distinct peaks are observed. For RRI1, AR(4) seems to be capable of representing the fundamental frequency. For RRI2, AR(10) which is the highest model order of all three trials, is also capable of representing the fundamental frequency. For RRI3, AR(7) is used. For all trials, and

especially for trial 3, the harmonics are not conserved when the PSDs are modelled. The AR models also seem to estimate the tails poorly. These estimations can help with the visualisations of the peaks periodogram, keeping in mind to not over-fit to the noise.

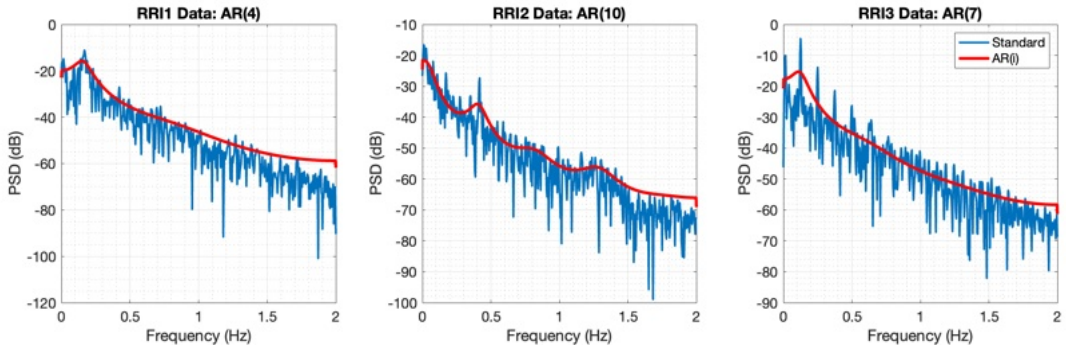


Figure 1.13: AR spectrum Estimate of each RRI Data

1.6 Robust Regression

The singular values of X and X_{noise} and the square error between them are plotted in Figure 1.14. Only the first 3 values are non-zero, the it's rank is 3. The squared error shows that the first 3 indexes maintains the low squared error, but the significant increase in error after this index implies that the true rank of X_{noise} will become hard to distinguish beyond the 3rd dimension.

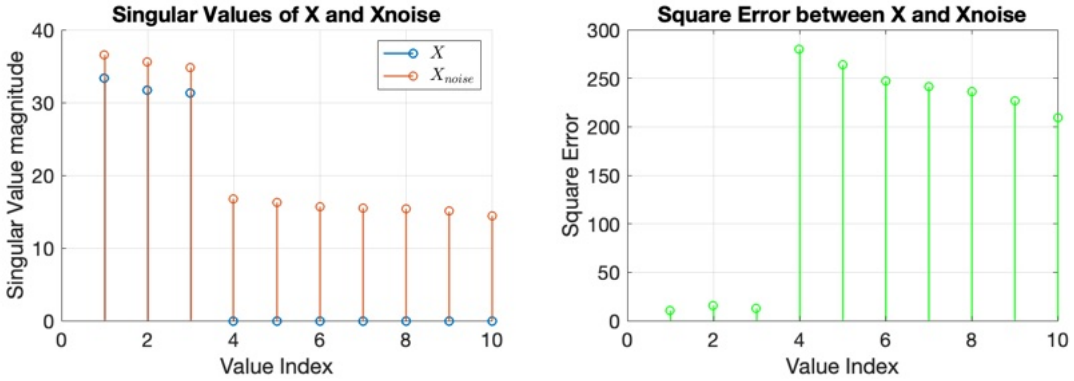


Figure 1.14: Singular values of X and Xnoise, ans the squared error

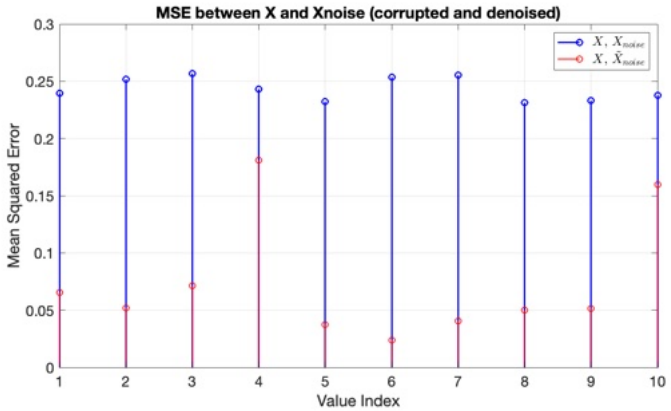


Figure 1.15: AR spectrum Estimate of each RRI Data

Principal component analysis (PCA) was performed on X_{noise} using only the first 3 components on X_{noise} to crease a low rank approximation $\tilde{X}_{noise}$. The MSE between X and X_{noise}, and X and $\tilde{X}_{noise}$

are plotted in Figure 1.15. As expected, the MSE between X and X_{noise} is significantly greater due to the non-zero singular values in subspace dimensions greater than 3.

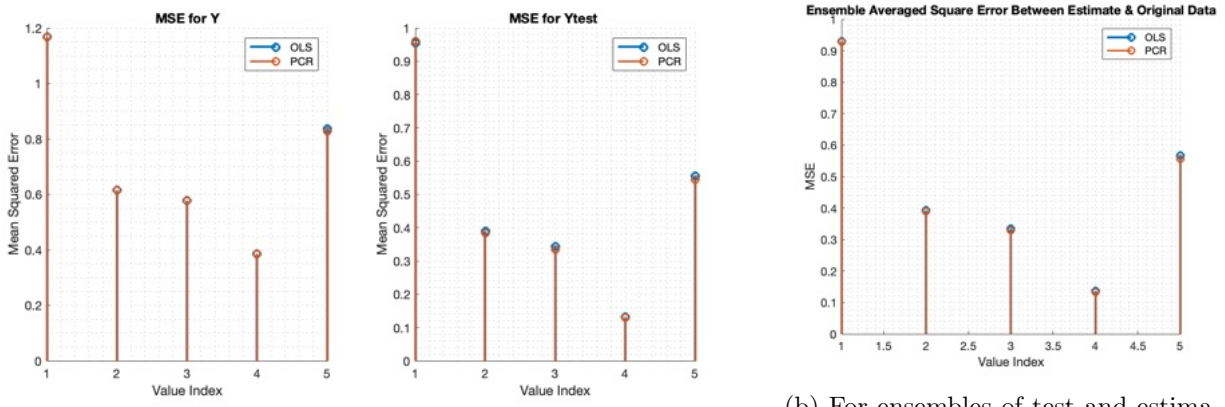
Deriving the output Y is done using the equation $Y = XB + N_Y$, where B is the estimated regressions coefficients obtained using either OLS or PCR.

$$\hat{B}_{OLS} = (X_{noise}X_{noise}^T)^{-1}X_{noise}^T Y \tag{16}$$

$$\hat{B}_{PCR} = V_{1:r}(\Sigma_{1:r})^{-1}U_{1:r}^T Y \tag{17}$$

where the SVD $X_{noise} = U\Sigma V^T$.
The estimate error between these outputs and original sample/test data is displayed in Figure 1.16. The MSE is not visibly different in the sample data. In the test data, the MSE is slightly larger for OLS, suggesting a propensity to overfitting to the noise.

The function regval was used to compare the effectiveness of PCR and OLS by generating realisations of test data Y and their estimates $\hat{Y}$. This allows for the calculation of MSE for OLS and PCR based methods using an ensemble to data. The MSE is plotted in Figure 1.16. We can see that PCR performs slightly better. However this marginal improvement in performance may not be too advantageous because PCA is much more computationally expensive for high ranks.



(a) Error for (Y, $\hat{Y}_{OLS}$, $\hat{Y}_{PCR}$), and (Y_{test},$\hat{Y}_{test-OLS}$,$\hat{Y}_{test-PCR}$)

(b) For ensembles of test and estimator data

Figure 1.16: Error analysis on data generated via OLS and PCR methods

2 Adaptive signal processing

2.1 The Least Mean Square (LMS) Algorithm

2.1.1 Identification of AR processes

The second order AR process can be expressed by:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n) \qquad \eta(n) \sim N(0, \sigma_\eta^2) \tag{18}$$

where the parameter values are $a_1 = 0.1$,$a_2 = 0.8$ and $\sigma_\eta^2 = 0.25$. We will be implementing adaptive linear prediction using the LMS algorithm. The inputs to the LMS filters are $x(n-1)$ and $x(n-2)$, which is presented as a vector input:

$$x(n) = [x(n-1), x(n-2)]^T \tag{19}$$

The correlation matrix based on the input vector is derived and written as an autocorrelation matrix:

$$R_{xx} = E[x(n)x(n)^T] = \begin{bmatrix} x(n-1)x(n-1) & x(n-1)x(n-2) \\ x(n-2)x(n-1) & x(n-2)x(n-2) \end{bmatrix} = \begin{bmatrix} r_{xx}(0) & r_{xx}(1) \\ r_{xx}(1) & r_{xx}(0) \end{bmatrix} \tag{20}$$

To calculate the values of the autocorrelation function, the general form for a AR(2) process is used. Note the linearity of the $E[]$ operator.

$$r_{xx}(k) = E[x(n)x(n-k)] = E[a_1x(n-1)x(n-k) + a_2x(n-2)x(n-k) + \eta(n)x(n-k)] \tag{21}$$
$$r_{xx}(k) = a_1E[x(n-1)x(n-k)] + a_2E[x(n-2)x(n-k)] + E[\eta(n)x(n-k)] \tag{22}$$

So the derived expressions are:

$$r_{xx}(0) = a_1r_{xx}(1) + a_2r_{xx}(2) + \sigma_\eta^2 \tag{23}$$
$$r_{xx}(1) = a_1r_{xx}(0) + a_2r_{xx}(-1) = a_1r_{xx}(0) + a_2r_{xx}(1) \tag{24}$$
$$r_{xx}(2) = a_1r_{xx}(1) + a_2r_{xx}(0) \tag{25}$$

where σ_η^2 is the variance of the noise and $E[\eta(n)x(n-k)] = 0$ when $k > 0$. Solving the simultaneous equations gives us $r_{xx}(0) = 25/27$ and $r_{xx}(1) = 25/54$.

$$R_{xx} = \begin{bmatrix} \frac{25}{27} & \frac{25}{54} \\ \frac{25}{54} & \frac{25}{27} \end{bmatrix} \tag{26}$$

To allow for LMS to converge to the mean, the range of values for time step μ must be:

$$0 < \mu < \frac{2}{\lambda_{max}} \tag{27}$$

where λ_{max} is the largest eigenvalue of the autocorrelation matrix R_{xx}. Using eigen-analysis, $\lambda_{max} = 1.389$, Hence $0 < \mu < 1.440$.

The LMS adaptive predictor was implemented and the squared prediction error was plotted along time using step sizes $\mu = 0.05$ and $\mu = 0.01$. This is then repeated for 100 realisations where the average realisation was obtained to form the learning curve (Figure 2.1).

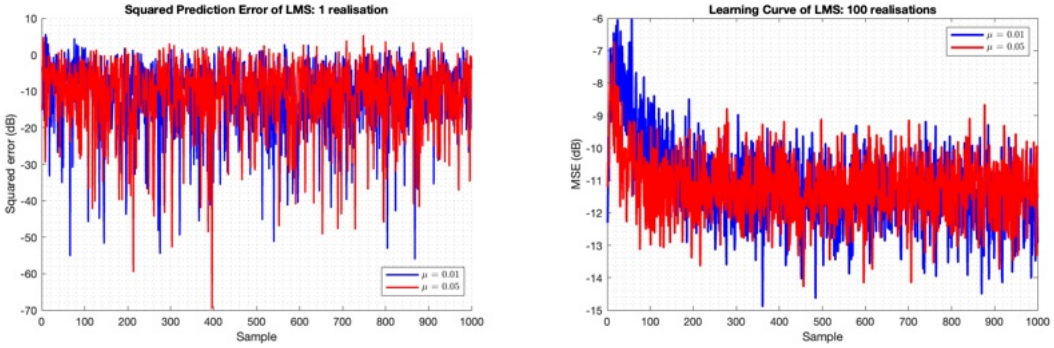


Figure 2.1: AR spectrum Estimate of each RRI Data

The theoretical and empirical misadjustment values are compared. The additional error power introduced by the adaptive filter can be quantified by the excess mean square error (EMSE) metric which measures the difference between the mean-square error introduced by adaptive filters and the minimum attainable mean square error of a Wiener filter. The corresponding misadjustment, M, is the ratio of the excess mean square error and the minimum mean square error. The empirical value can be calculated as follows.

$$MSE = \lim_{n\to\infty} E\{e^2(n)\} = \sigma_\eta^2 + EMSE \tag{28}$$
$$M = \frac{EMSE}{\sigma_\eta^2} = \frac{MSE}{\sigma_\eta^2} - 1 \tag{29}$$
$$\tag{30}$$

The theoretical value is defined as the ratio of ESME to minimum MSE, but for small step sizes it can be approximated as:

$$M_{LMS} \approx \frac{\mu}{2} Tr\{R_{xx}\} = \frac{\mu}{2} \times 1.851 \tag{31}$$

where $Tr\{.\}$ is the matrix trace operator. The time-averaging over steady-state is done using coefficients from $n = 500$ to the end. Hence the results are presented in the table below.

μ	M_{emp}	M_{theo}
0.01	0.0080368	0.00926
0.05	0.053822	0.0463

There is a slight mismatch between the pairs of values, which may be due to the approximation done in the theoretical derivation, the finite signal length, and the estimation of the time-index for steady-state. Although the empirical values are less than the theoretical values (i.e. less excess errors), these values are not reliable because there is large variance across different runs.

The steady state values of the adaptive filter coefficients for the step-sizes were estimated by averaging the coefficients across 100 independent trials, as shown in Figure 2.2. The step size of $\mu = 0.01$ does a better job at converging into the true coefficients (0.8 and 0.1) than step size $\mu = 0.05$. Therefore, using a larger step size decreases estimation performance, but increases rate of convergence. Hence a trade off exists between convergence speed and steady state error.

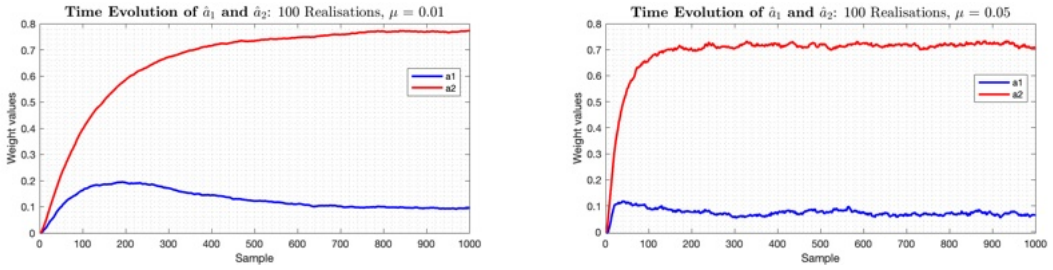


Figure 2.2

2.1.2 The Leaky LMS Algorithm

Deriving the LMS coefficient update for w(n) that minimizes the cost function results in the leaky LMS equation. So starting from the beginning of the derivation, the cost function to promote stabilisation is given as:

$$J_2(n) = \frac{1}{2}\left(e^2(n) + \gamma ||w(n)||_2^2\right) \tag{32}$$

where the leakage constant γ is introduced. The error term can be rewritten in terms of the signal $x(n)$ and output $y(n)$. Note that the vectors are replaced with appropriate expressions to reduce confusion caused by multiple brackets.

$$J_2(n) = \frac{1}{2}\left((\mathbf{y}(n) - \mathbf{w}(n)^T\mathbf{x}(n))^T(\mathbf{y}(n) - \mathbf{w}(n)^T\mathbf{x}(n)) + \gamma \mathbf{w}(n)^T\mathbf{w}(n)\right) \tag{33}$$

$$= \frac{1}{2}\left((Y - W^T X)^T(Y - W^T X) + \gamma W^T W\right) \tag{34}$$

Then the gradient of the cost function is derived by differentiating w.r.t. the weights:

$$\nabla J_2(n) = \frac{1}{2}\left(-X_n^T(Y_n - W_n^T X_n) - X_n(Y_n - W_n^T X_n)^T + 2\gamma W_n\right) \tag{35}$$

$$= \frac{1}{2}\left(-2X_n^T(Y_n - W_n^T X) + 2\gamma W_n\right) \tag{36}$$

$$= \left(-X_n^T(Y_n - W_n^T X_n) + \gamma W_n\right) \tag{37}$$

The gradient descent update to derive the minimum of the cost function is defined as:

$$w(n) = w(n) - \mu \nabla J_2(n) \tag{38}$$

$$W_{n+1} = W_n - \mu \nabla_w J_2(n) \tag{39}$$

Substituting it gives:

$$W_{n+1} = W_n - \mu\left(-X_n^T(Y_n - W_n^T X_n) + \gamma W_n\right) \tag{40}$$

$$= (1 - \gamma\mu)W_n + \mu X_n^T(Y_n - W_n^T X_n) \tag{41}$$

$$w(n) = (1 - \gamma\mu)w(n) + \mu x(n)e(n) \tag{42}$$

hence the leaky LMS equation is derived.

The leaky LMS algorithm was implemented for the same AR(2) process using different leaning rates, and the results are provided below (Figure 2.3 and 2.4). the filter weights fail to converge to a steady-state that has values equal to the coefficients. Furthermore, the larger the leakage coefficient γ, the greater the bias in the converged coefficient estimates.
To provide theoretical backing for this observation, the relationship between the leakage coefficient γ and the bias in coefficient estimates can be explained by referring to the Wiener-Hopf solution.

$$w = R_{xx}^{-1} p \tag{43}$$

where $R_x x$ is the autocorrelation matrix and p is the cross-correlation vector between input $x(n)$ and output $y(n)$. When the autocorrelation matrix has zero-valued eigenvalues, the LMS filter may become unstable because $R_x x$ is no longer positive definite and may not be invertible. In order to resolve this, the leakage factor is introduced.

$$w = (R_{xx} + \gamma I)^{-1} p \tag{44}$$

As $\gamma \to 0$, this expression converges to the ideal Wiener-Hopf solution, and bias is reduces to zero.

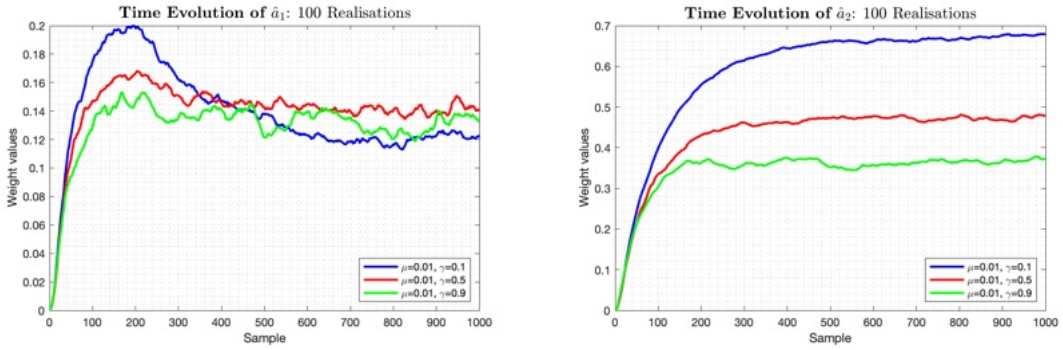


Figure 2.3: Evolution of filter weights in LMS for learning rates $\mu = 0.01$

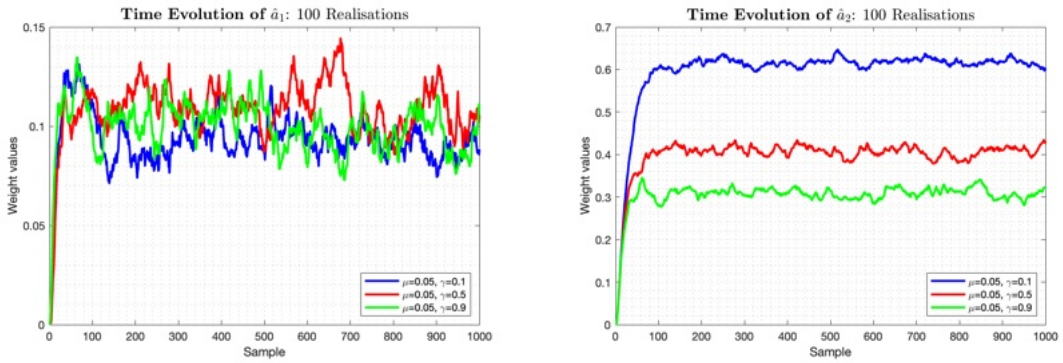


Figure 2.4: Evolution of filter weights in LMS for learning rates $\mu = 0.05$

2.2 Adaptive Step Sizes

For a normal LMS algorithm, a trade off exists between convergence speed and steady state error due to the constant step size. So gradient adaptive step-size (GASS) algorithms are designed with this in mind, the step size minimisations are gradient based where μ is large when converging and small at steady state.
Identification of a real-valued MA(1) system, as notated below, was used to compare the performance of the gradient adaptive step-size (GASS) algorithms.

$$x(n) = 0.9\eta(n-1) + \eta(n) \qquad \eta \sim N(0, 0.5) \tag{45}$$

The GASS algorithms are Benveniste, Ang and Farhang, and Matthews and Xie. The weight errors are plotted as shown in Figure 2.5. All 3 algorithms perform better than a constant μ.
. They all converge to the steady state weight error much faster and, among the GASS algorithms, Benveniste has the lowest steady state error, perhaps because it has the highest computational complexity. Although GASS algorithms converge much faster, the steady state error achieved is higher than that for the constant time step of $\mu = 0.01$, as observed in Figure 2.6. When the GASS algorithms are initialised with $\mu_{initGASS} = 0$, Benveniste also has the greatest convergence speed. In the order of the highest to lowest convergence speed, the HASS algorithms are Benveniste, Ang and Farlang, and Matthews and Xie.

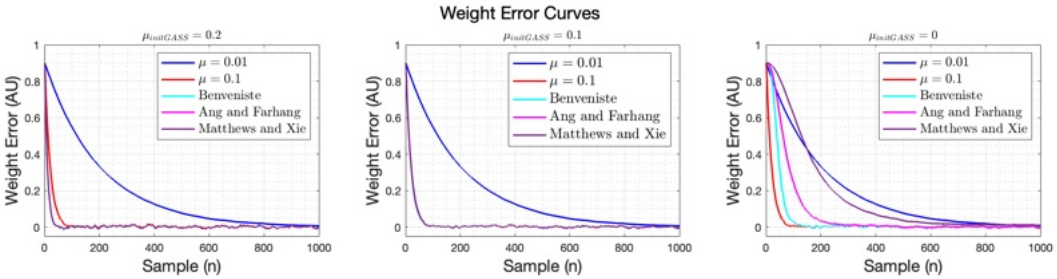


Figure 2.5: Weight Error curve with GASS step-size initialised $\mu_{initGASS}$

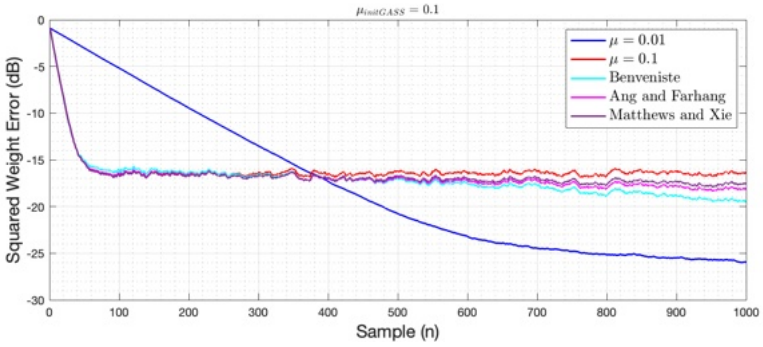


Figure 2.6: A closer look at Squared Weight Error curve for $\mu_{initGASS} = 0.1$

To verify that the update equation,

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

based on a posteriori error $e_p(n) = d(n) - x^T(n)w(n+1)$, is equivalent to the Normalized Least Mean Squares (NLMS) algorithm, we first multiply both sides by $x(n)^T$.

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

$$= \mathbf{x}(n)^T\mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n)^T\mathbf{x}(n) \tag{48}$$

$$= \mathbf{x}(n)^T\mathbf{w}(n) + \mu e_p(n)||\mathbf{x}(n)||^2 \tag{49}$$

Then we substitute $\mathbf{x}(n)^T\mathbf{w}(n+1)$ into the posteriori error.

$$e_p(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1) = d(n) - \left(\mathbf{x}(n)^T\mathbf{w}(n) + \mu e_p(n)||\mathbf{x}(n)||^2\right) \tag{50}$$

The error for a normal LMS algorithm is $e(n) = d(n) - \mathbf{x}(n)^T\mathbf{w}(n)$, hence the equation can be rearranged.

$$e_p(n) = e(n) - \mu e_p(n)||\mathbf{x}(n)||^2 \tag{51}$$

$$e_p(n)\left(1 + \mu||\mathbf{x}(n)||^2\right) = e(n) \tag{52}$$

$$e_p(n) = e(n)\left(\frac{1}{1 + \mu||\mathbf{x}(n)||^2}\right) \tag{53}$$

Back to the update equation we substitute $e_p(n)$:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\left(\frac{1}{1 + \mu||\mathbf{x}(n)||^2}\right)e(n)\mathbf{x}(n) = \mathbf{w}(n) + \left[\frac{\mu}{1 + \mu\mathbf{x}(n)^T\mathbf{x}(n)}\right]e(n)\mathbf{x}(n) \tag{54}$$

It is equivalent to the provided NLMS algorithm provided when $\beta = 1$ and $\epsilon = 1/\mu$.

2.2.1 The Generalised Normalised Gradient Descent (GNGD)

The GNDG algorithm is a normalised version of LMS, where the adaptive gain is achieved through the regularisation term ϵ instead of μ in the cost function. Figure 2.7 presents a comparison between the two techniques. For the plot, GASS-Benveniste's was implemented with the parameters $\rho = 0.002$ and $\mu = 0.1$. GNGD was implemented with the parameters $\rho = 0.005$ and $\mu = 1$. After experimenting with different parameter values, GNDG converges faster when the stability is achieved with the correct parameter values. However, the steady-state weight error for Benveniste's is lower. The Benveniste's algorithm contains matrix multiplications, which is more computationally complex compared to the vector additions and inner products required in GNGD. So the choice between the two algorithms would have to depend on computational resource and the importance of steady-state accuracy.

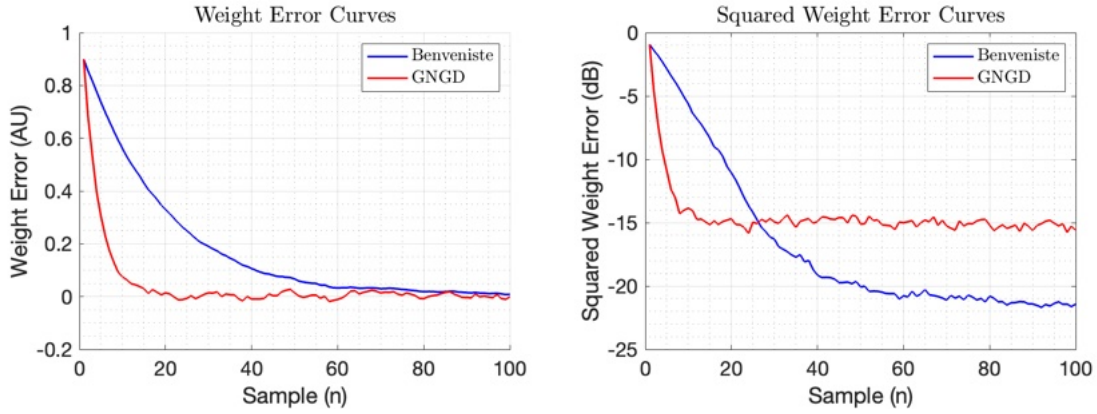


Figure 2.7: GNGD and Benveniste weight error and squared weight error curves

2.3 Adaptive Noise Cancellation

2.3.1 Adaptive Line Enhancer (ALE)

The minimum value for the delay Δ that may be used in the ALE for a filter length of $M > 1$ can be derived as follows. Starting by expanding $MSE = E\{(s(n) - \hat{x}(n))^2\}$, we get:

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

$$= E\{\eta^2(n)\} + E\{(x(n) - \hat{x}_\Delta(n))^2\} + 2E\{\eta(n)(x(n) - \hat{x}_\Delta(n))\} \tag{56}$$

where the time delay underscore is the prediction at a certain time delay. The first term does not have any dependency on Δ and the second term will be zero when Δ is appropriately chosen. So minimising will only need to be done on the last term.

$$MSE_{min} = \min_{\Delta} E\left\{\eta(n)(x(n) - \hat{x}_{\Delta}(n))\right\} = \min_{\Delta} \left(E\left\{\eta(n)x(n)\right\} - E\left\{\eta(n)\hat{x}_{\Delta}(n)\right\}\right) \tag{57}$$

Assuming that the signal $x(n)$ is independent of the noise $\eta(n)$, and using $\eta(n) = v(n) + 0.5v(n-2)$, $\hat{x}(n,\Delta) = \mathbf{w}^T\mathbf{u}(n,\Delta)$ and $\mathbf{u}(n,\Delta) = [s(n-\Delta), ..., s(n-\Delta-M+1)]^T$ the equation can be reduced to:

$$MSE_{min} = \min_{\Delta} E\left\{\eta(n)\hat{x}_{\Delta}(n)\right\} = \min_{\Delta} E\left\{(v(n) + 0.5v(n-2))\mathbf{w}^T\mathbf{u}(n,\Delta)\right\} \tag{58}$$

$$= \min_{\Delta} E\left\{(v(n) + 0.5v(n-2)) \sum_{m}^{M-1} w_n s(n-\Delta-m)\right\} \tag{59}$$

This is further expanded using $s(n) = x(n) + \eta(n)$

$$MSE_{min} = \min_{\Delta} E\left\{(v(n) + 0.5v(n-2)) \sum_{m}^{M-1} w_n \left(x(n-\Delta-m) + \eta(n-\Delta-m)\right)\right\} \tag{60}$$

$$= \min_{\Delta} E\left\{(v(n) + 0.5v(n-2)) \sum_{m}^{M-1} w_n \left(\eta(n-\Delta-m)\right)\right\} \tag{61}$$

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

$$= \min_{\Delta} \sum_{m}^{M-1} w_n E\Big\{(v(n) + 0.5v(n-2))\left(v(n-\Delta-m) + 0.5v(n-\Delta-m-2)\right)\Big\} \tag{63}$$

where the term $x(n-\Delta-m)$ was removed because it is uncorrelated to the noise. Since $v(n)$ is a white Gaussian noise, in order to minimise the derived equation, $\Delta > 2$ must hold hence the minimum delay is 3. To justify the minimum, ALE was implemented using the LMS algorithm across 100 iterations, at filter order M=5 and $\mu = 0.01$. In Figure 2.8, you can see a significant decrease in MSPE from $\Delta = 2$ to $\Delta = 3$ and visually in the plot.

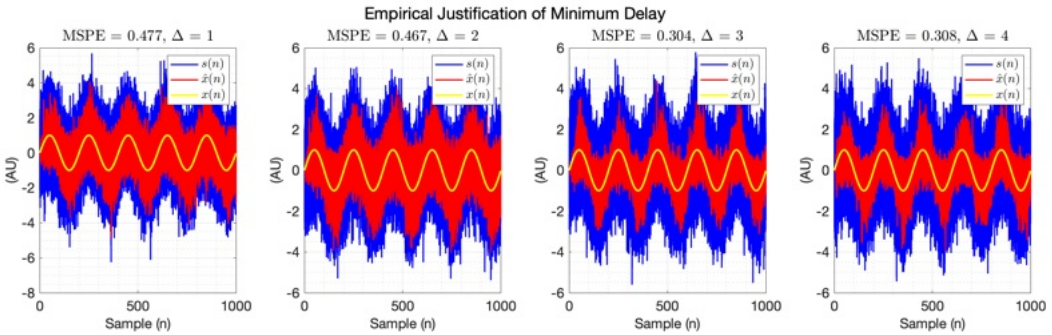


Figure 2.8

1000 samples of s(n) was generated and the ALE-LMS was used to estimate x(n) for filter orders $M = 5, 10, 15, 20$. Then the MSPE for a range of filter oders are calculated. The results are presented in Figure 2.9. Firstly, increasing filter order increases MSPE values across all delays by the same amount (the curves are shaped the same). As filter order increases, the MSPE decreases to a minimum at $M = 5$ before increasing steadily. $M = 5$ would be the best if computational resources are not scarce. Although $M = 3$ would be a good choice too because the difference in MSPE is marginal.

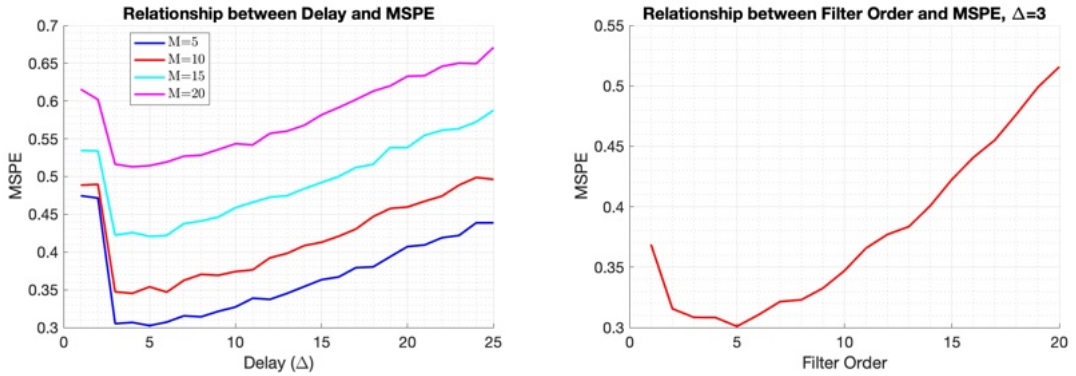


Figure 2.9: The relationship between delay and MSPE, ans filter order and MSPE

2.3.2 Adaptive Noise Cancellation (ANC)

The Adaptive Noise Cancellation configuration (ANC) configuration was implemented and the outcome was compared to ALE using MSPE as the measure of performance. ALE estimates the noise present in the signal, $\eta(n)$, using the secondary noise measurement $\epsilon(n)$ and then the clean signal $x(n)$ is formed by subtracting $\eta(n)$ from $s(n)$. For this run, $\epsilon(n) = 1.5\eta(n) + 0.1$ and filter order $M = 5$ was used, and the output is provided in Figure 2.10.

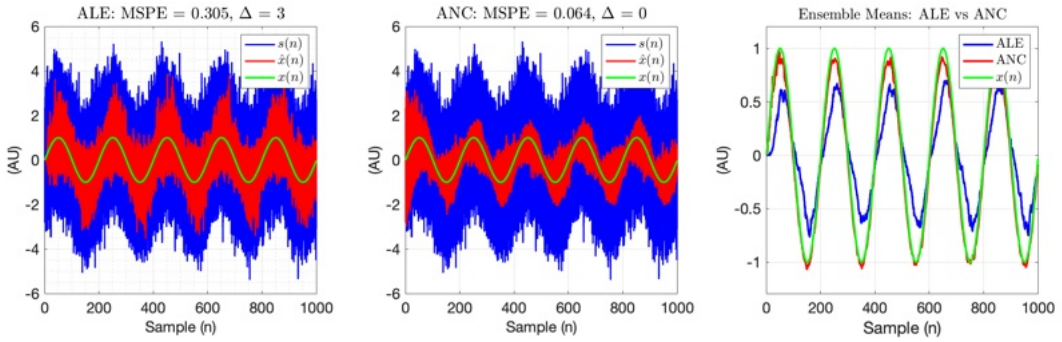


Figure 2.10: 100 realisations of the input signal $s(n)$ and the predicted signal $\hat{x}(n)$, with $\epsilon(n) = 1.5\eta(n) + 0.1$ for ANC

ANC takes a longer time to converge and have a lower MSPE result when the values of the reference noise $\epsilon(n)$ allows for it. A bad noise reference was used to illustrate this point (Figure 2.11). So ALE is the better choice if response time is a priority and while ANC would be much better for when quality out output is importance, where more effort is required for the selection of noise reference.

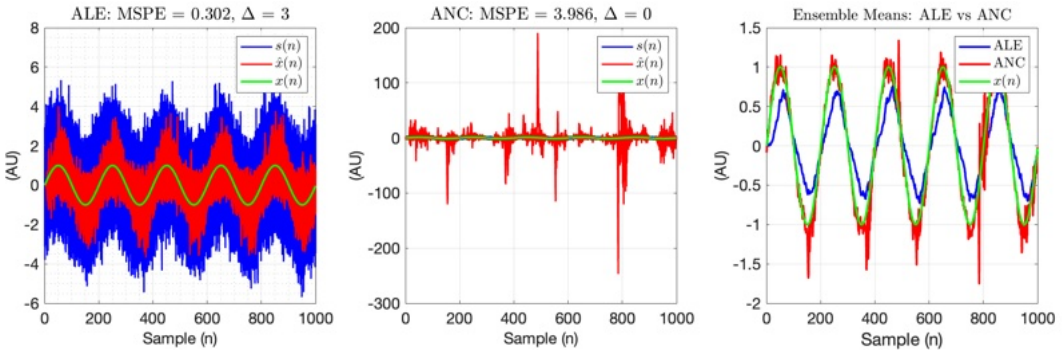


Figure 2.11: 100 realisations of the input signal $s(n)$ and the predicted signal $\hat{x}(n)$, with $\epsilon(n) = 3.5\eta(n) + 0.1$ for ANC

Using real-world data, ANC was applied to the POz electrode of an EEG recording. The goal is to remove mains noise (50Hz) without resulting in information loss in the remaining frequency bands.

For the ANC, the reference noise input was set as a unit-amplitude 50Hz sinusoid, noise-corrupted by WGN with power 0.005. For the spectrogram, a Hanning window has the length of 6144 (for higher frequency resolution instead of time resolution), with a overlap of 2/3 between windows, and an FFT size for each frame of POz data of 5×4096.

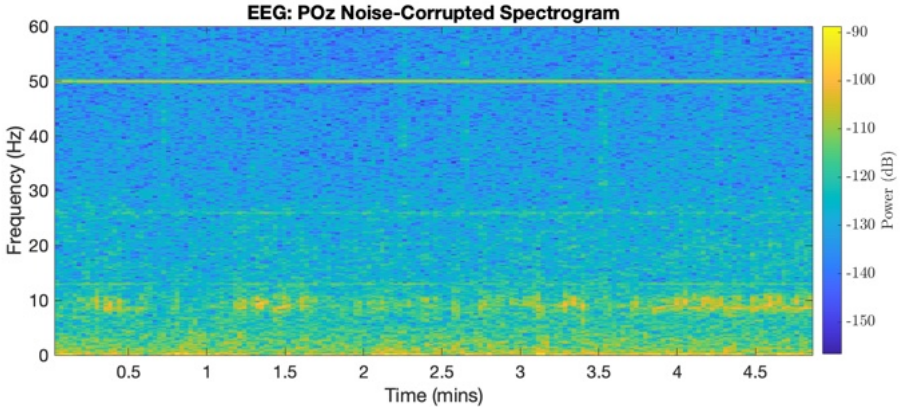


Figure 2.12: Spectrogram of the POz EEG signal.

The effect of filter order and learning rate (M and μ) on the performance of the ANC, at removing the $50Hz$ noise, was investigated as displayed in Figure 2.13. Increasing μ increases the frequency range of ANC denoising due to the larger jumps in filter weights. Increase in M also increases the denoising performance, but when it is too high (e.g. $M > 20$), the power in the $50Hz$ band become unnaturally low. So, looking at the spectograms, the optimal choice of parameters would be around $M = 10$ and $\mu = 0.001$.

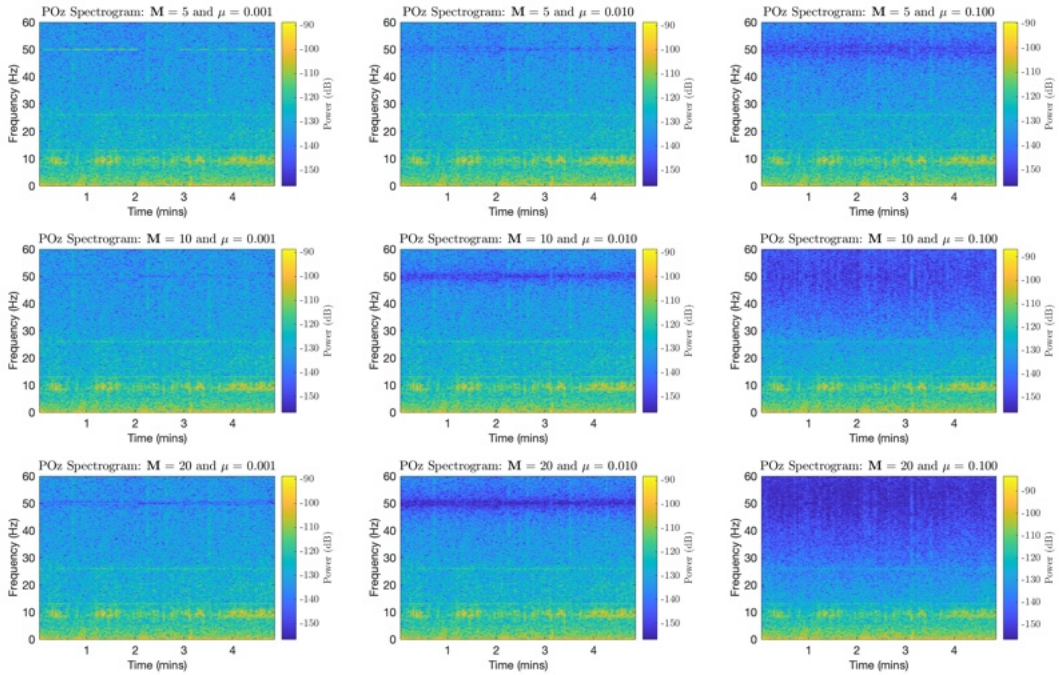


Figure 2.13: Spectrograms after ANC denoising of an EEG signal, for $M = [5, 10, 20]$ and $\mu = [0.001, 0.01, 0.1]$

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

A first-order widely-linear-moving-average process, WLMA(1), was generated and driven by circular white Gaussian noise:

$$y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-1) \qquad x \sim N(0,1) \tag{64}$$

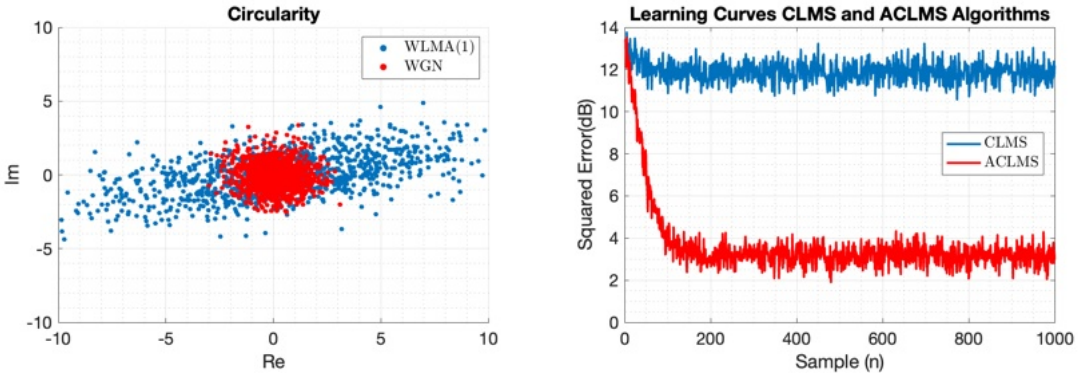


Figure 3.1: Left: plot to show circularity , Right: prediction performance of CLMS and ACLMS

The resulting signal is non-circular (Figure 3.1). CLMS and ACLMS were implemented for this generated signal and their absolute squared error was plotted in dB, displayed in Figure 3.1. As expected, ACLMS has a lower steady-state error compared to CLMS. The CLMS algorithm does not include consideration for second order statistics in the data, specifically the pseudocovariance (how its real and imaginary components are correlated). For circular data, the pseudocovariance matrix is all zero, hence due to the nature of the signal, ACLMS can handle these relationships when learning.

3.1.1 Bivariate wind data

The circularity plots for the bivariate data that is loaded into complex arrays in the form:

$$v[n] = v_{east}[n] + jv_{north}[n] \tag{65}$$

and the centres of mass were calculated and plotted with a yellow marker (Figure 3.2). The circularity coefficients are $\rho_{low} = 0.62366$, $\rho_{med} = 0.45431$, and $\rho_{high} = 0.15919$, where $\rho = 0$ is perfectly circular and $\rho = 1$ is fully non-circular. So we can conclude that increasing speed increases circularity.

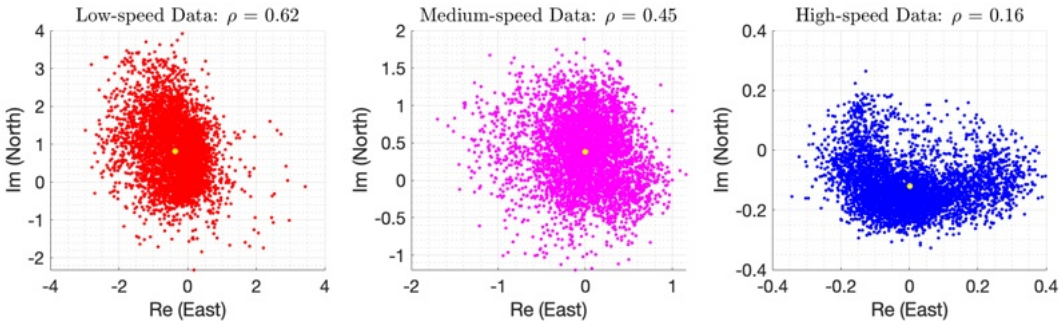


Figure 3.2: Circularity plots where $\rho_{low} = 0.62366$, $\rho_{med} = 0.45431$, and $\rho_{high} = 0.15919$

The CLMS and ACLMS filters were configured in the prediction setting, and their performance based on different filter lengths were explored (Figure 3.3). For low speeds, ACLMS performs much better than CLMS, all the way until filter order of 23. For medium speeds, ACLMS perform better until filter order 9, from when CLMS produces a lower MSPE. This is also observed for high speeds. Overall,

ACLMS gets worse and CLMS gets better at performing as speed increases, whihc is expected because the signal becomes more circular. However at the minimum MSPE, ACLMS is still the better choice for all wind speeds.

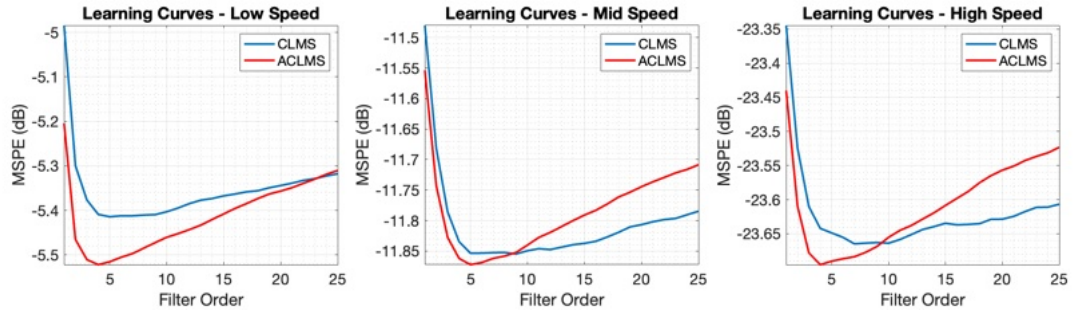


Figure 3.3

3.2 Adaptive AR Model Based Time-Frequency Estimation

3.2.1 Power spectrum of a frequency modulated (FM) signal

The frequency modulated (FM) signal, expressed below, where $\eta(n)$ is the circular complex-valued white noise with zero mean and variance $\sigma_\eta^2 = 0.05$.

$$y(n) = e^{j(\frac{2\pi}{fs}\phi(n))} + \eta(n) \tag{66}$$

The signal was generated and visualised in Figure 3.4, showing non-stationarity due to varying frequency against the time-axis.

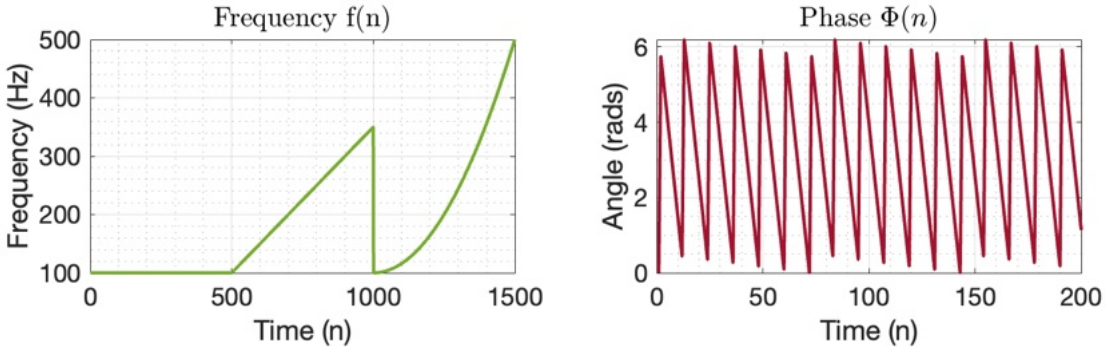


Figure 3.4: Showing the expected frequencies and phase for the signal

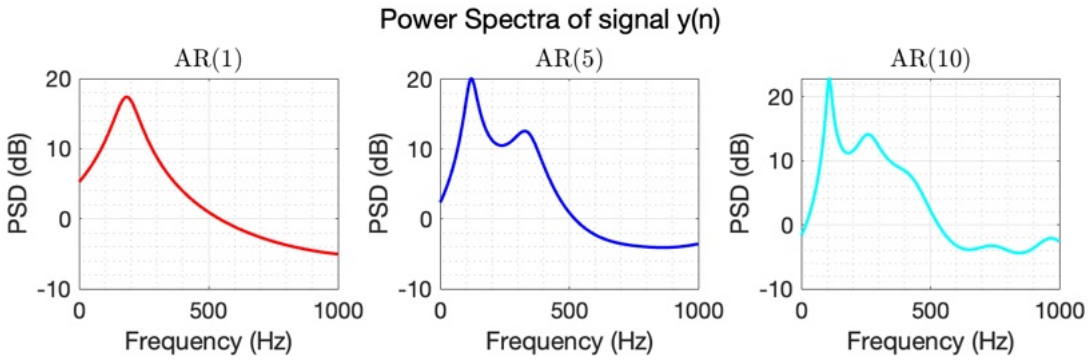


Figure 3.5

The power spectrum of the signal was plotted in Figure 3.5 using the AR(1) coefficient obtained using the *aryule* function for the complete signal of length 1500. *aryule* assumes stationarity, so the model

fails to capture accurate frequencies, even up to AR(10). Looking at Figure 3.4 and the AR(10) spetra, there is a peak at $100Hz$ that could be from the frequency range of $1Hz$ to $500Hz$. However it is not clear where the peak at $250Hz$ relates to. Perhaps it represents the frequency range of $1Hz$ and $500Hz$ in time index 1000 to 1500. At AR(1), the peak is around $176Hz$, the average for the generated signal.

3.2.2 CLMS based spectrum estimate

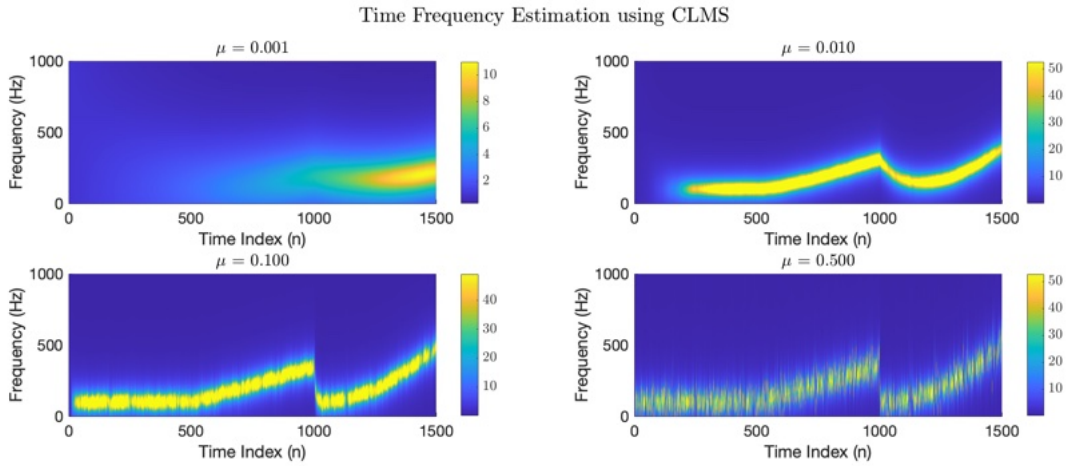


Figure 3.6: CLMS with different μ performed on a FM signal

The CLMS algorithm was implemented to estimate the AR coefficients of the signal. This should work better than the *aryule* function because CLMS is an adaptive method that can handle small non-linearities and the signal is quite circular. These assumptions were correct as shown in Figure 3.6, as we are able to make out the time-dependent frequencies in the signal. Note that the clarity of the result depends on learning rate μ. When it is too small ($\mu = 0.001$), the coefficients take too long to converge and cannot adapt fast enough to changes in frequency. When it is too large ($\mu = 0.5$), The frequency oscillates prominently as the steps taken between coefficient updates are too large which provides large variance in estimation. So an optimal μ would have to be considered.

3.3 A Real Time Spectrum Analyser Using Least Mean Square

We have a signal $y(n)$ that is estimated using a linear combination of N harmonically related sinusoids:

$$\hat{y}_n = \sum_{k=0}^{N-1} w(k) e^{\frac{j2\pi kn}{N}} \tag{67}$$

where $\hat{y}_n$ is the estimate of the signal $y(n)$ and $w(k)$ are the unknown weights (the Fourier coefficients). $y(n)$ is the form of a vector can be expressed as:

$$\underbrace{\begin{bmatrix} \hat{y}(n) \\ \hat{y}(n) \\ \vdots \\ \hat{y}(n) \end{bmatrix}}_{\hat{y}} = \underbrace{\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{\frac{j2\pi(1)(1)}{N}} & \cdots & e^{\frac{j2\pi(1)(N-1)}{N}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{\frac{j2\pi(N-1)(1)}{N}} & \cdots & e^{\frac{j2\pi(N-1)(N-1)}{N}} \end{bmatrix}}_{\mathbf{F}} \underbrace{\begin{bmatrix} w(0) \\ w(1) \\ \vdots \\ w(N-1) \end{bmatrix}}_{w} = \mathbf{F}w \tag{68}$$

where $\mathbf{F}$ is the matrix of harmonically related sinusoids and w is the vector of unknown weights (the Fourier coefficients). To solve the least squares problem, we start with the cost function:

$$J(w) = ||y - \hat{y}||^2 = ||y - \mathbf{F}w||^2 = (y - \mathbf{F}w)^H (y - \mathbf{F}w) \tag{69}$$

The cost function is then minimised using the coefficients.

$$\left.\frac{\delta J(w)}{\delta w}\right|_{w=w^*} = \left.\frac{\delta\left(\mathbf{y}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}w - w^H\mathbf{F}^H\mathbf{y} + w^H\mathbf{F}^H\mathbf{F}w\right)}{\delta w}\right|_{w=w^*} = 0 \tag{70}$$

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

Solving for w^* gives

$$w^* = \left(\mathbf{F}^H\mathbf{F}\right)^{-1}\mathbf{F}^H\mathbf{y} \tag{72}$$

Hence the matrix $\mathbf{F}$ as well as the expression $(\mathbf{F}^H\mathbf{F})^{-1}$must be invertible. $\mathbf{F}$ is invertible because it is full-rank, due to the orthogonality of basis vectors (i.e. sinusoids of different frequencies in each column). Using this knowledge, we can further simplify the expression:

$$w^* = \left(\mathbf{F}^H\mathbf{F}\right)^{-1}\mathbf{F}^H\mathbf{y} = (N\mathbf{I})^{-1}\mathbf{F}^H\mathbf{y} = \frac{1}{N}\mathbf{F}^H\mathbf{y} \tag{73}$$

By comparing it to the discrete Fourier transform (DFT) formula and its inverse:

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi nk}{N}} \tag{74}$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi nk}{N}} \tag{75}$$

We can see the relationship:

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k)e^{\frac{j2\pi kn}{N}} = \mathbf{FY} \tag{76}$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\frac{2\pi nk}{N}} = \mathbf{F_{DFT}X} \tag{77}$$

If we let $\mathbf{X} = \mathbf{Y}$, we the the following expression for $\hat{y}$ base on the DFT,

$$\mathbf{F} = N\mathbf{F_{DFT}} \tag{78}$$

So the signal can be represented using the Fourier coefficients of a DFT:

$$\hat{y}_n = \sum_{k=0}^{N-1} w(k)e^{\frac{j2\pi kn}{N}} = \mathbf{F}w^* = N\mathbf{F_{DFT}}w^* \tag{79}$$

Serving as the relationship between the least squares solution and DFT.

3.3.1 Fourier transform in terms of the change of basis and projections.

As previously mentioned, a key property of the DFT is the mutual orthogonality of its basis vectors, where each column represents a sinusoid at a different frequency. It becomes mutually orthonormal when $\frac{1}{N}$ is included and the inner products of any two basis functions (of different frequencies) is always zero. A signal $x(n)$ in the time domain is typically represented in the standard basis, where each basis vector corresponds to a discrete-time impulse at a particular index. The DFT transforms this representation into a new basis consisting of complex exponentials (sinusoids of different frequencies). The DFT computes the best least-squares approximation of $x(n)$ in the space spanned by the sinusoidal basis functions, where each Fourier coefficient in $\mathbf{F}$ represents the strength of the projection of $x(n)$ onto the corresponding frequency component.

3.3.2 DFT-CLMS for FM signal

DFT-CLMS was implemented on the FM signal generated previously and the magnitude of the weight vector $w(n)$ at every time instant is plotted to create a time-frequency diagram as shown in Figure 3.7. The parameters for this plot are $\mu = 1$ and $\gamma = [0, 0.01, 0.05, 0.5]$. The algorithm responds very well to the time-varying frequency, but "bleeding" is observed in the spectrum and this effect depends on the leakage γ. This is due to the fact that when $\gamma = 1$ (i.e. when there's no leakage) DFT-CLMS does not "forget" past information fast enough to adapt to the varying frequency. Using leakage increases rate of adaptation by "throwing away" information that is no longer important, although an appropriate value should be used that gives the best balance between forgetting and signal distortion. In this case, by observing the plots in Figure 3.7, we can determine the optimal value to be in the range $0.01 < \gamma < 0.05$.

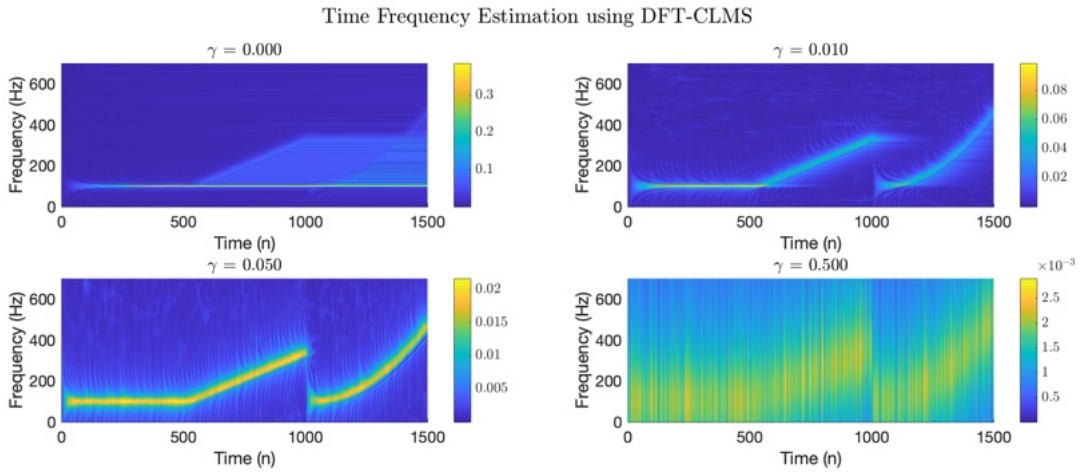


Figure 3.7: DFT-CLMS performed on FM signal

3.3.3 DFT-CLMS for the EEG signal POz

The DFT-CLMS algorithm was implemented for a segment of length 1200 from the EEG signal POz. The time-frequency spectrum is displayed in Figure 3.8, with parameters $\mu = 1$ and $\gamma = [0, 0.01, 0.05, 0.5]$ again. The non-zero leakage coefficients result in distortion and artefacts in the spectrum, to the point that we cannot identify the important frequency bands present. For $\gamma = 0$, the $13Hz$ SSVEP is visible and the first harmonic ($26Hz$) can also be seen, but higher harmonics are not observable. The power-line noise at $50Hz$ is also not visible. There is no point using any amount of leakage at all because the 1200 samples of EEG can be approximated as stationary.

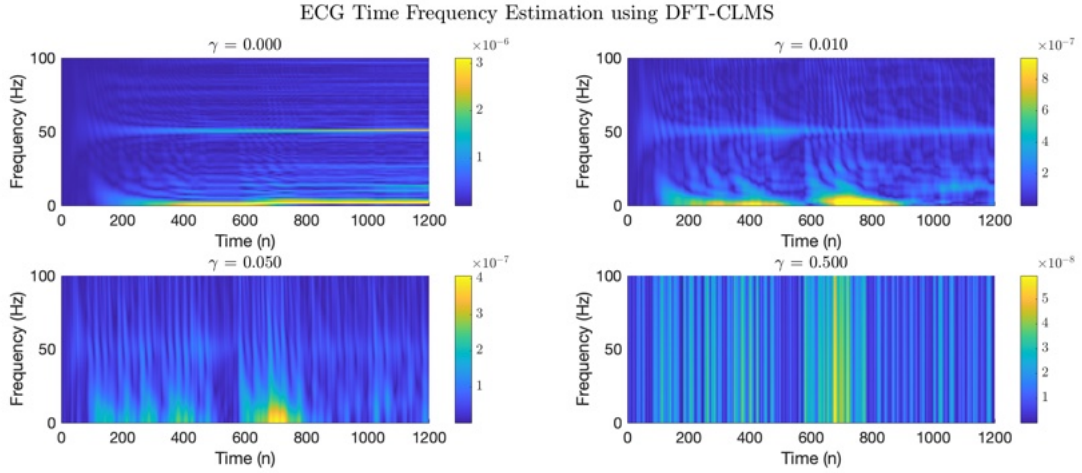


Figure 3.8: DFT-CLMS performed on EEG POz data

4 From LMS to Deep Learning

4.1 Neural Networks for Prediction

4.1.1 LMS algorithm

The time-series provided was loaded and made to be zero-mean. The LMS algorithm was implemented and the one-step ahead prediction was plotted along with the original signal in Figure 4.1, using step size $\mu = 0.00005$ and model order 4.
Looking close at Figure 4.1, we see the transient period in about the first 200 samples where the filter has yet to adapt to the signal, so there is a great mismatch. Towards the end of the signal prediction, the error is much lower compared to that of the overall signal, $MSE = 23.38dB$ vs $MSE = 16.03dB$, and the prediction gain is also much higher, at $Rp = 9.44dB$ vs $Rp = 5.20dB$. This shows an improvement in prediction.

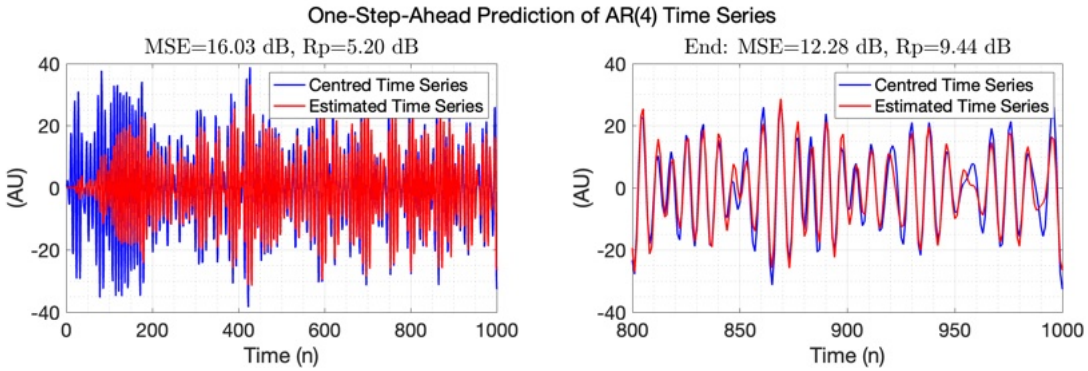


Figure 4.1: Centred signal and one-step ahead prediction for when using LMS

4.1.2 Dynamical perceptron: activation function

In order to cater to the non-linearities present in the signal, an improvement is added in the form of the activation function to the prediction output.

$$\hat{y}(n) = \tanh\left(w(n)^T x(n)\right) \tag{80}$$

and the weights are updated while keeping this non-linear function in mind.

$$w(n+1) = w(n) + \mu\left(1 - \tanh^2\left(w(n)^T x(n)\right)\right) e(n)x(n) \tag{81}$$

where the identity $\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$ is used. This new algorithm is coined as the dynamical perceptron.

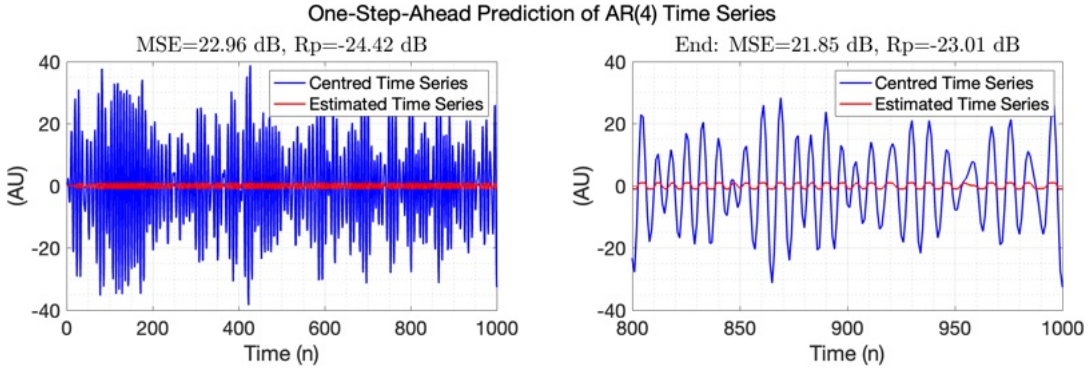


Figure 4.2: Centred signal and one-step ahead prediction for when using unscaled activation function

The algorithm was implemented and the one-step ahead prediction was plotted in Figure 4.2. We can see that although the predicted signal is in phase with the original signal, the scale of the prediction

is too small. This is due to the the tanh function where the amplitudes are capped at 1 and -1. So we will move on to the next step to alleviate this problem.

4.1.3 Dynamical perceptron: scaling

As an improvement, the scaling factor is added before the tanh function in the form of the alpha parameter, α. The weight update is altered to accommodate for this change.

$$\hat{y}(n) = \alpha \tanh\left(w(n)^T x(n)\right) \tag{82}$$

$$w(n+1) = w(n) + \mu\alpha\left(1 - \tanh^2(w(n)^T x(n))\right) e(n)x(n) \tag{83}$$

where the learning rate $\mu\alpha$ is now scale appropriately. The next step is to find the optimal value of α.

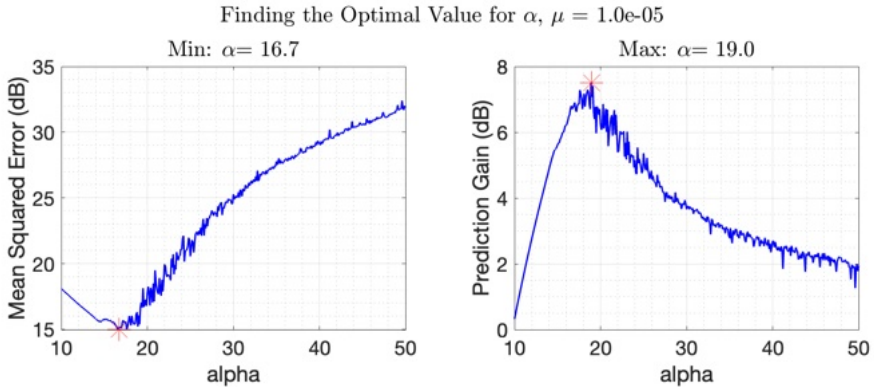


Figure 4.3: Selecting optimal value for α when $\mu = 1e$-5

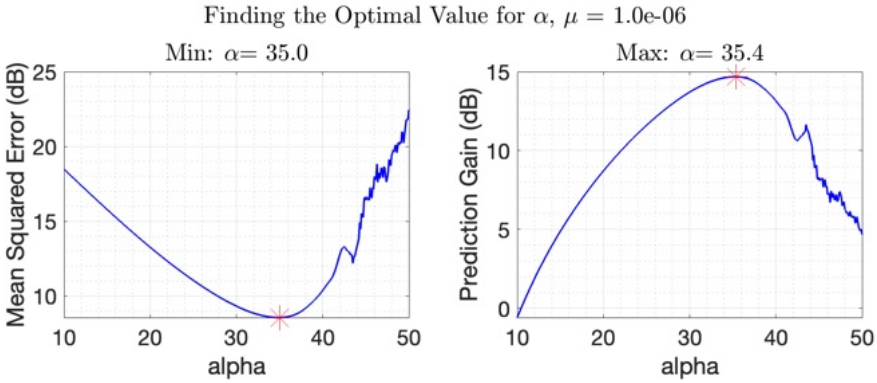


Figure 4.4: Selecting optimal value for α when $\mu = 1e$-6

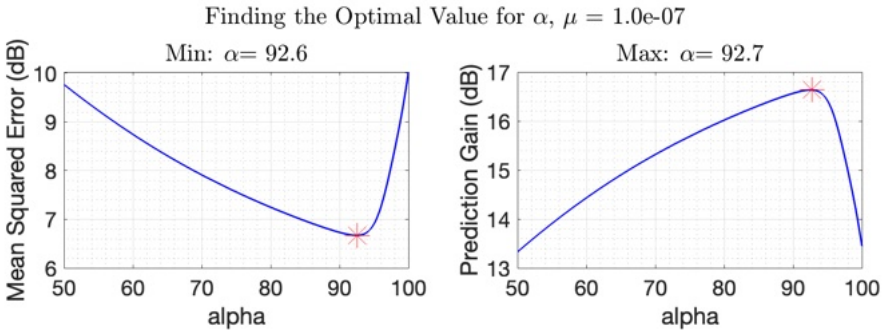


Figure 4.5: Selecting optimal value for α when $\mu = 1e$-7

Figures 4.3, 4.4 and 4.5 shows the plots of MSE against a range of α and R_p (prediction gain) for the purpose of investigating the optimal α where MSE is the lowest and R_p is the highest.

For step sizes 1e-5 and 1e-6, Figures 4.3 and 4.4, the trajectory of α is quite unstable, i.e. small changes in from optimal value give high variance or unpredictable performance, and the MSE and Rp values are worse than that of the smallest step size investigated. So we are using μ =1e-7 where the optimal value chosen is $\alpha = 92$.
The zero-mean signal and the new prediction using the selected parameters is plotted in Figure 4.6, and algorithm now performs better than the LMS algorithm in the previous section, with $MSE = 6.67dB$ and $Rp = 16.63dB$ overall and $MSE = 4.32dB$ and $Rp = 18.08dB$ at the end. The weight convergence is also visualised in Figure 4.7.

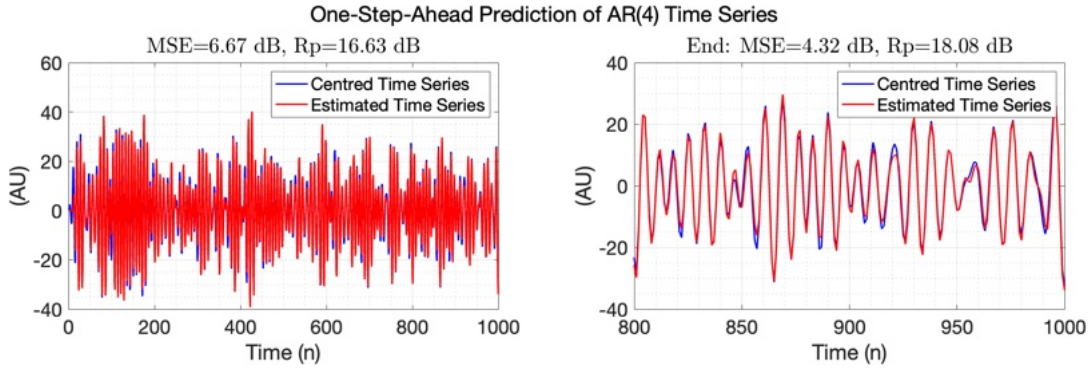


Figure 4.6: Centred signal and one-step ahead prediction for when using scaled activation function

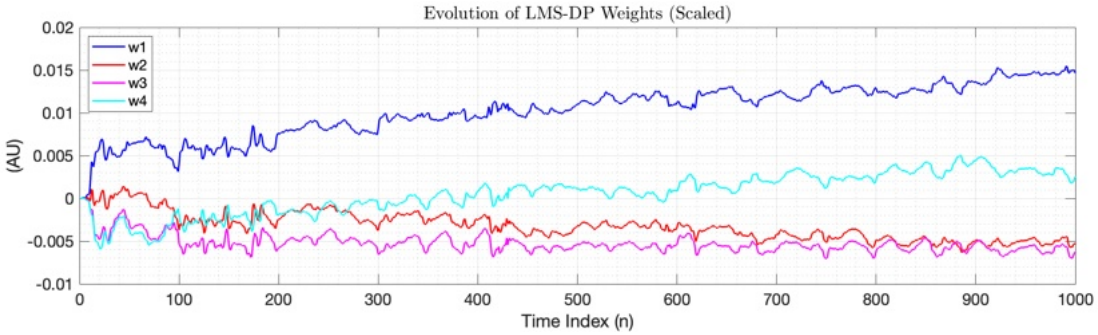


Figure 4.7: Weight convergence for LMS-DP with after scaling

4.1.4 Dynamical perceptron: adding bias

In real life settings, the input is usually biased, instead of zero-mean. The activation function in this case would become $\phi(\mathbf{w}^T\mathbf{x} + b)$, where b is the bias. In the code, this was done using the augmented input method as described.

$$\hat{y}(n) = \phi\left(W_{aug}X_{aug}\right) = \phi\left(\begin{bmatrix} w_0(n) & w_1(n) & \dots & w_M \end{bmatrix} \begin{bmatrix} 1 \\ x(n-1) \\ \vdots \\ x(n-M) \end{bmatrix}\right) \tag{84}$$

where M is the filter order ans the resulting estimate would look like: $\hat{y}(n) = \phi(\mathbf{w}^T\mathbf{x} + 1)$ when $bias = 1$.
The non-linear one-step ahead prediction with $bias = 1$ was plotted in Figure 4.8. The same parameters as before was used, which are μ =1e-7 and $\alpha = 92$. The output error did not change much compared to the one predicted without bias. It converges similarly to the unbiased output in Figure 4.6, where the prediction is better at the end of the signal, and the weights also converge similarly by comparing Figure 4.7 and 4.9. Hence it could be concluded that the LMS-DP algorithm works well even if there is a bias.

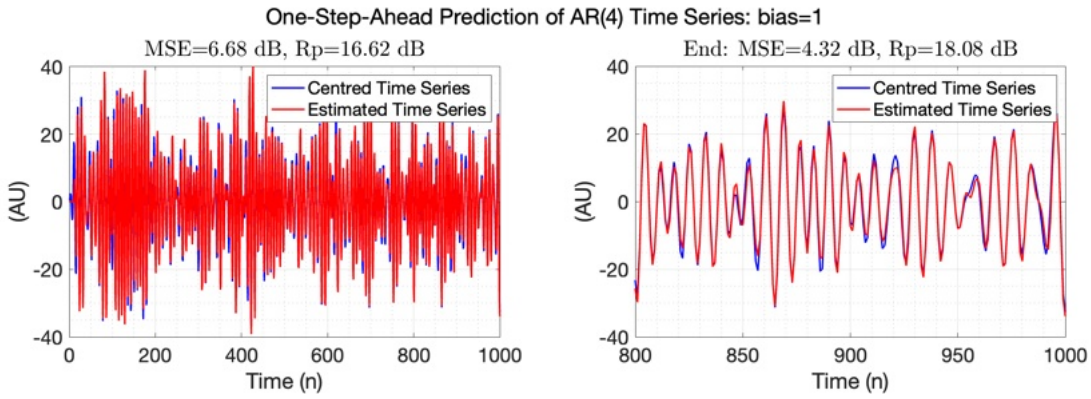


Figure 4.8: Centred signal and one-step ahead prediction for when using scaled activation function with bias=1

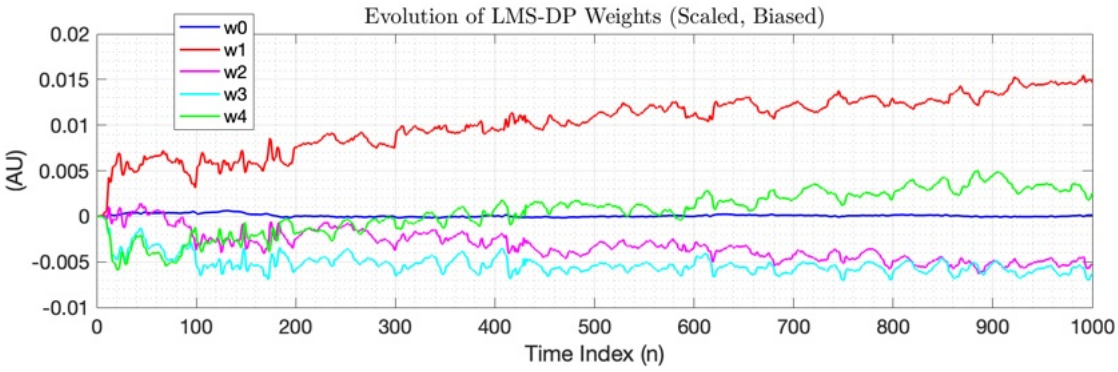


Figure 4.9: Weight convergence for LMS-DP with bias=1

The conclusion was stress tested with a higher DC offset of $bias = 10$ and the results are displayed in Figure 4.10. There is a slight increase in MSE and decrease in prediction gain Rp, but it still performs better overall compared to the normal linear LMS algorithm. So our conclusion is right.

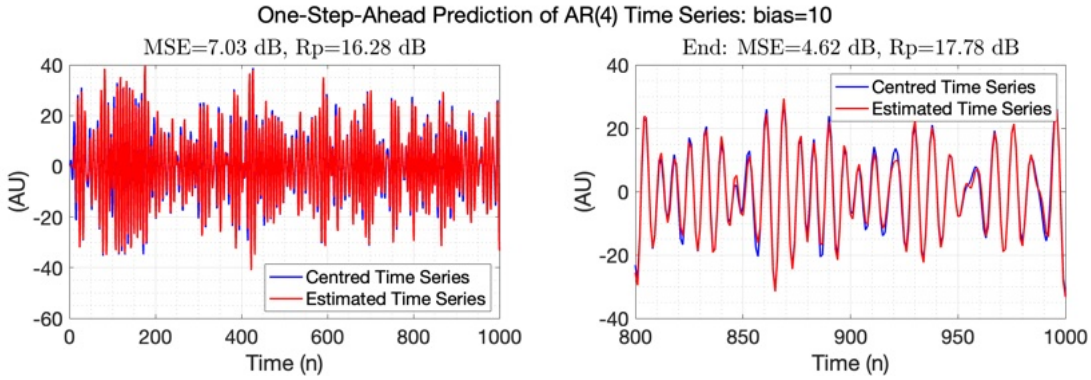


Figure 4.10: Centred signal and one-step ahead prediction for when using scaled activation function with bias=10

4.1.5 Dynamical perceptron: pre-trained weights

Using the biased settings, the weights were pre-trained by over-fitting to the first 20 samples, across 100 epochs. In this process, the optimal alpha was also selected (Figure 4.11). Then the weights are used to initialised the LMS-DP algorithm.

The final signal prediction is shown in Figure 4.12. Looking the the MSEs and prediction gains,

the performance is even better and with the expected improvements at the end of the signal, this algorithm outperforms all algorithms we have investigated up to now. The weight evolution in Figure 4.13 shows no convergence and stable values, which means that they are already converged before this algorithm is used, which is the result of the training that happened prior.

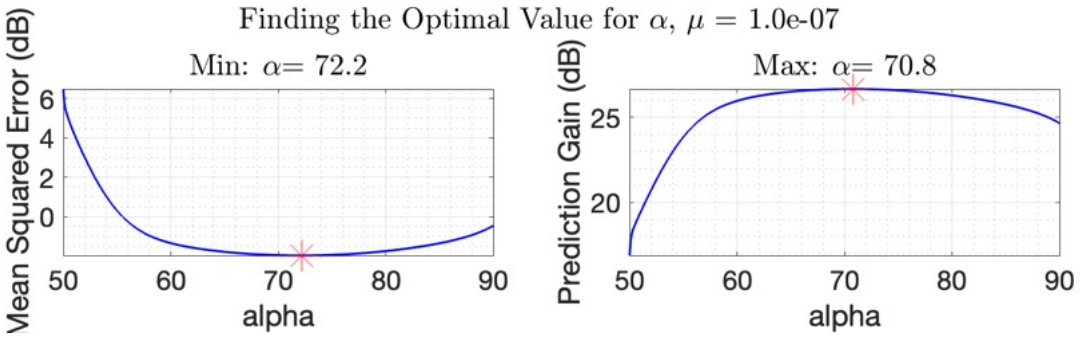


Figure 4.11: Selecting optimal value for α

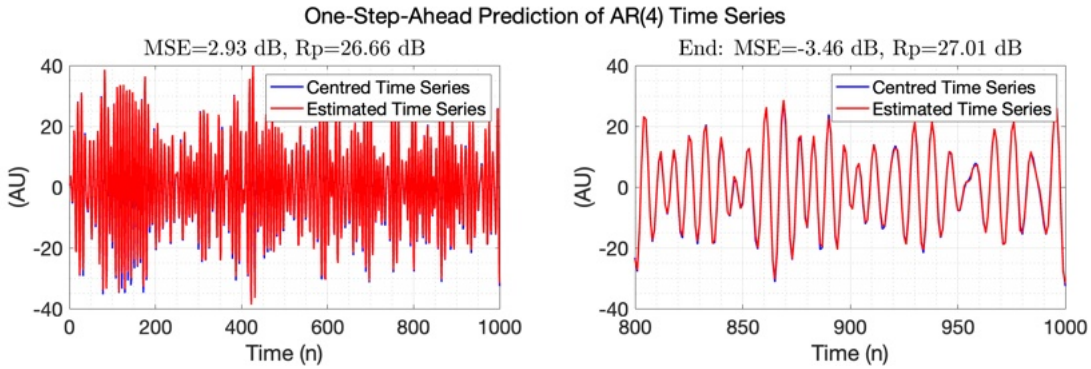


Figure 4.12: Centred signal and one-step ahead prediction for when using pre-trained weights and scaled activation function with bias=1

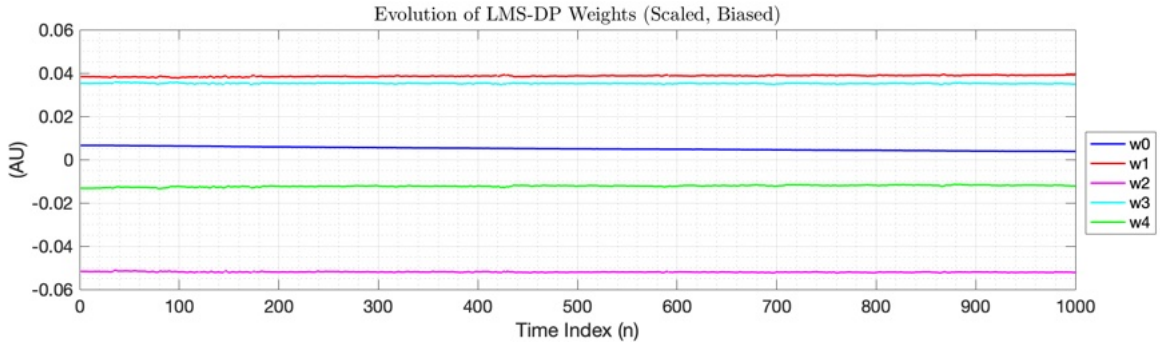


Figure 4.13: Weight convergence for LMS-DP after pre-trained

In all the sections up to this point, the optimal value for alpha has to be manually selected. A solution to this extra step is to use some form of adaptive update that would converge the parameter to its optimal value.

4.1.6 The Backpropagation Algorithm

Deep networks consist of multiple layers of simple dynamical perceptrons that are stacked together to form layers. To train these networks, we use the backpropagation algorithm, which efficiently updates the model's weights to minimize error.

The steps in the Backpropagation Algorithm includes:

1. **Forward Pass**, where the input is passed through the layers where non-linear weights are applied to get the output.
2. **Loss computation**, where the difference (usually MSE) between the the predicted output and target signal.
3. **Backward Pass** (Gradient Computation using Chain Rule) so that the error signals are propogated backwards.
4. **Weight Update** for all the layers (Gradient Descent)

In the following explanations the matrix notations used are *Input* $\mathbf{X}$, *Hidden Layer predictions* $\mathbf{A}_l$, *Output Layer prediction* $\mathbf{A}_L$, *Weights* $\mathbf{W}$, and *bias* $\mathbf{b}$. In the forward propagation, the activations are computed for the first, hidden and output layers:

$$\mathbf{A}_1 = \phi(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1) = \phi(\mathbf{Z}_1) \tag{85}$$

$$\mathbf{A}_l = \phi(\mathbf{W}_l\mathbf{A}_{(l-1)} + \mathbf{b}_l) = \phi(\mathbf{Z}_l) \tag{86}$$

$$\mathbf{A}_L = \phi(\mathbf{W}_L\mathbf{A}_{(L-1)} + \mathbf{b}_L) = \phi(\mathbf{Z}_L) \tag{87}$$

The loss $J = MSE(\mathbf{A}_L, \mathbf{Y})$ (for example) is calculated. Then the backward pass involves calculating the cost gradient using partial derivatives and chain rule, called the Gradient Descent method. Taking one pathway for a simple 4-layer neural network, we can calculate how much the error J effects the weight in the first layer:

$$\frac{\partial J}{\partial w^1} = \frac{\partial J}{\partial a^3}\frac{\partial a^3}{\partial z^3}\frac{\partial z^3}{\partial a^2}\frac{\partial a^2}{\partial z^2}\frac{\partial z^2}{\partial a^1}\frac{\partial a^1}{\partial z^1}\frac{\partial z^1}{\partial w^1} \tag{88}$$

And this is done for all the weights in all the layers and the process is repeated at the next prediction.

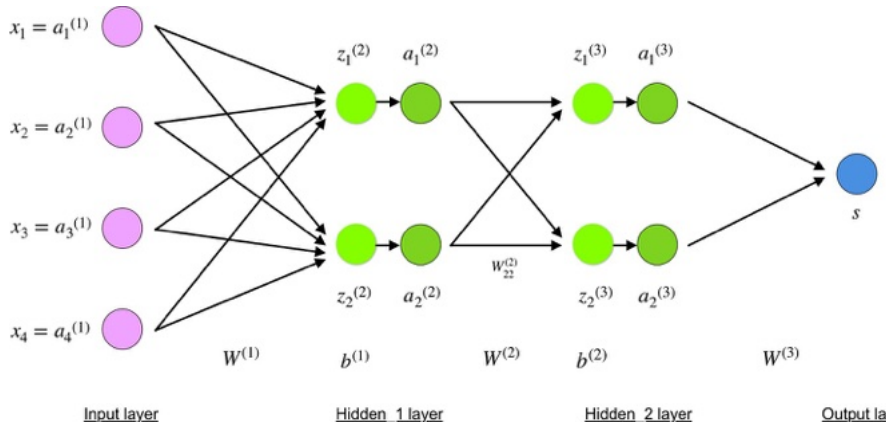


Figure 4.14: Illustrating Back Propagation for a simple 4-layer neural network

4.1.7 Deep network with default parameters

The deep neural network (DNN) was trained with default parameters (4 hidden layers, 20,000 epochs, learning rate of 0.01 and noise power equal to 0.05). Figure 4.15 (upper plot) shows the sinusoid components that are present when preparing the input signal before non-linearisation $x[n]$. Figure 4.15 (lower plot) demonstrates actual signal sample as an output from a highly non-linear model $y[n] = \phi(x[n])$.

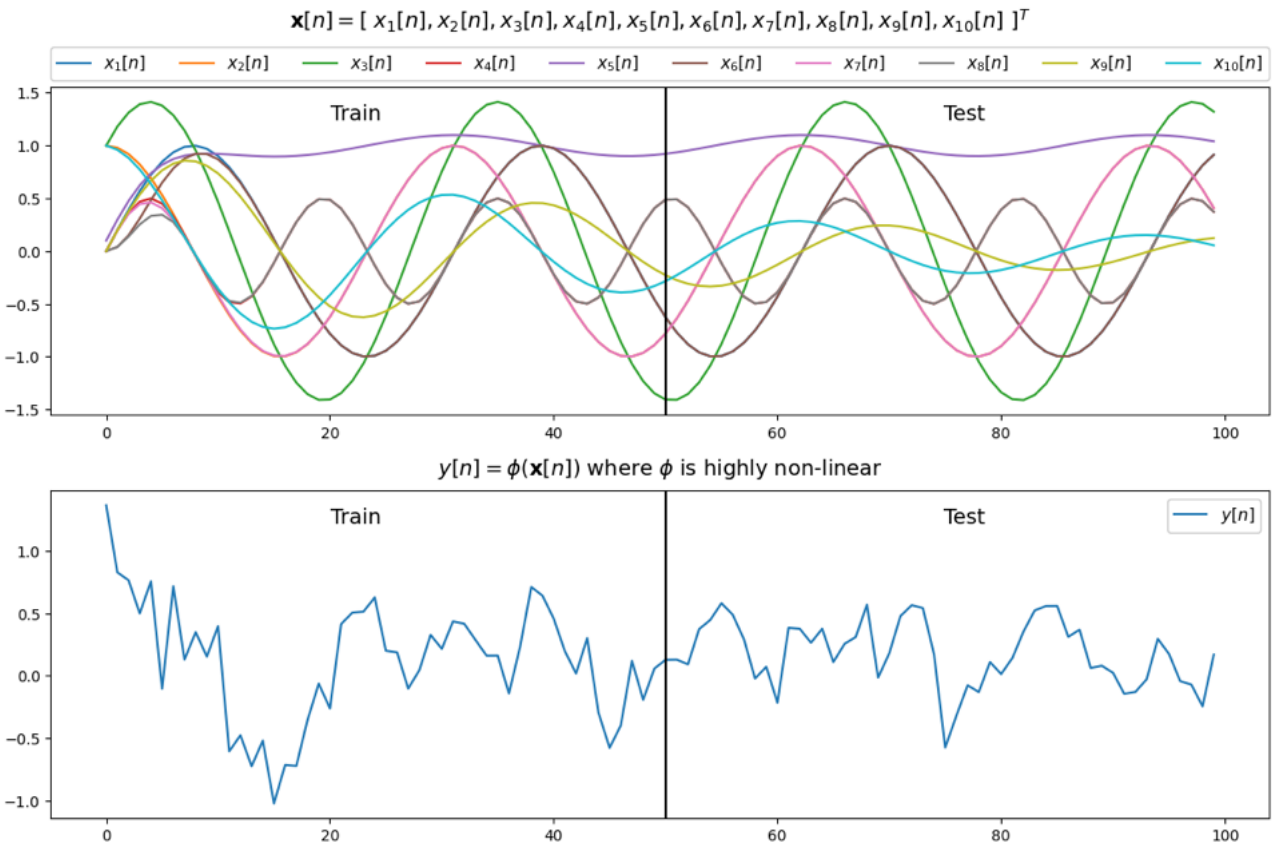


Figure 4.15: Plots showing the components of the test signal

The prediction signals are visualised in Figure 4.16 and their respective loss functions are plotted in Figure 4.17.

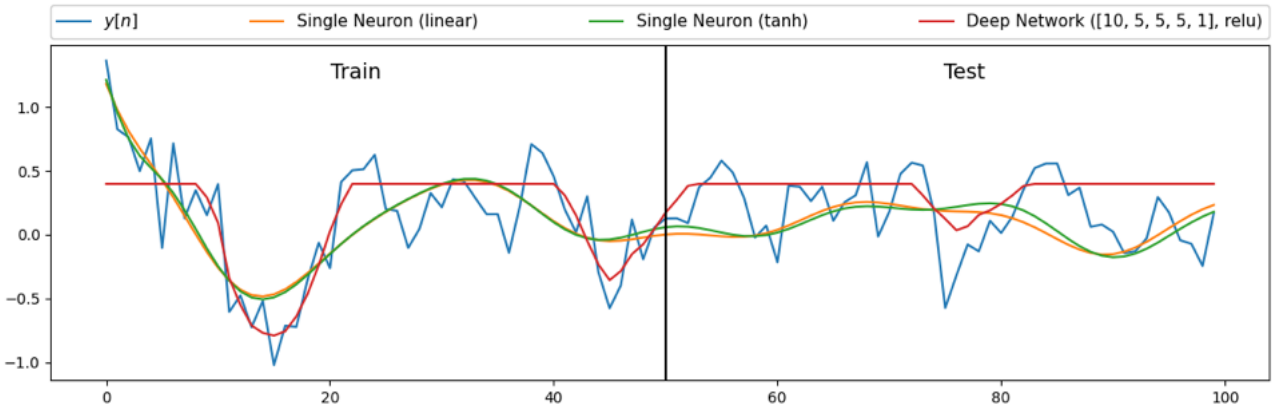


Figure 4.16: Prediction from different models

The three models we are comparing include: a Linear Dynamical Perceptron, Tanh-based Dynamical Perceptron (same as the previous section) and a 4-hidden-layer DNN with the Rectrified Linear Unit function (ReLu). ReLU is piecewise linear, where the gradient is 1 for positive inputs (fast learning) and 0 for negative inputs (dead neurons). It avoids the vanishing gradients problem while reducing computational cost. The performances were tested and gauged using the Loss functions during training and testing (Figure 4.18).

For the simple neurons (linear and tanh) they converge much faster compared to the DNN, and their test loss is very slightly, if not negligible, less than training loss. For the DNN, it takes more epochs to converge and, upon convergence, the overall error is less than that for the single neurons.

However, the DNN seems to have larger loss values during testing compared to the training.

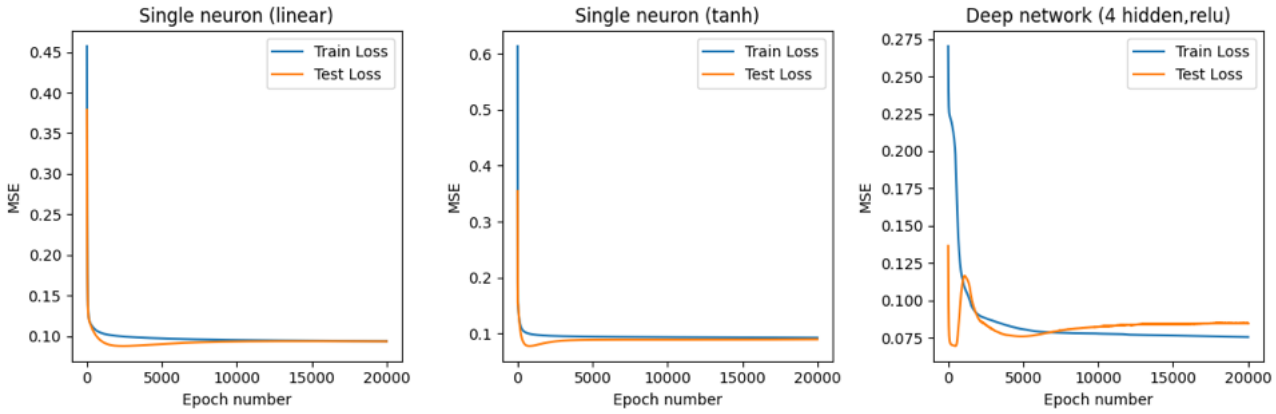


Figure 4.17: Default noise power=0.05

4.1.8 Different values of noise power

The DNN was, again, trained with different noise powers. The loss functions for noise power of 0.5 and 0.05 are displayed in Figures 4.18 and 4.19.

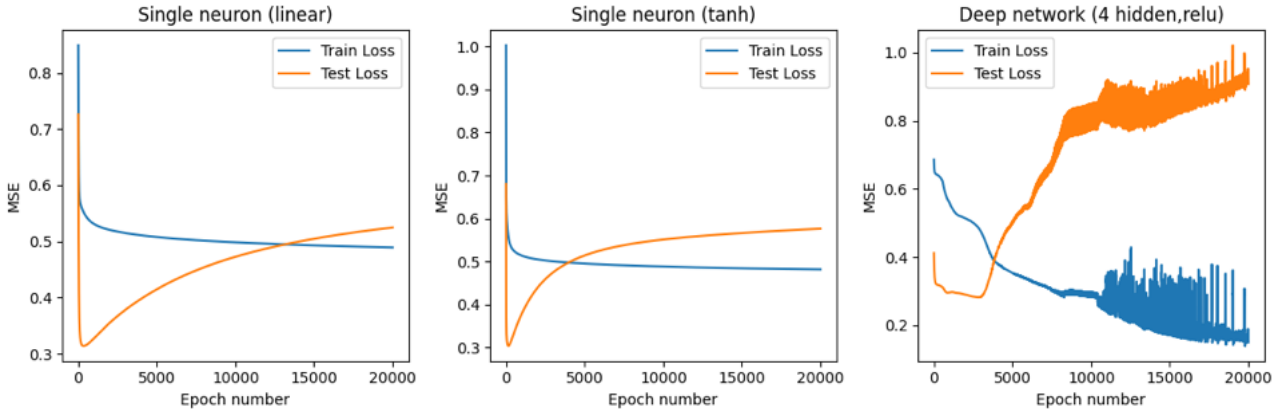


Figure 4.18: Larger noise power=0.5

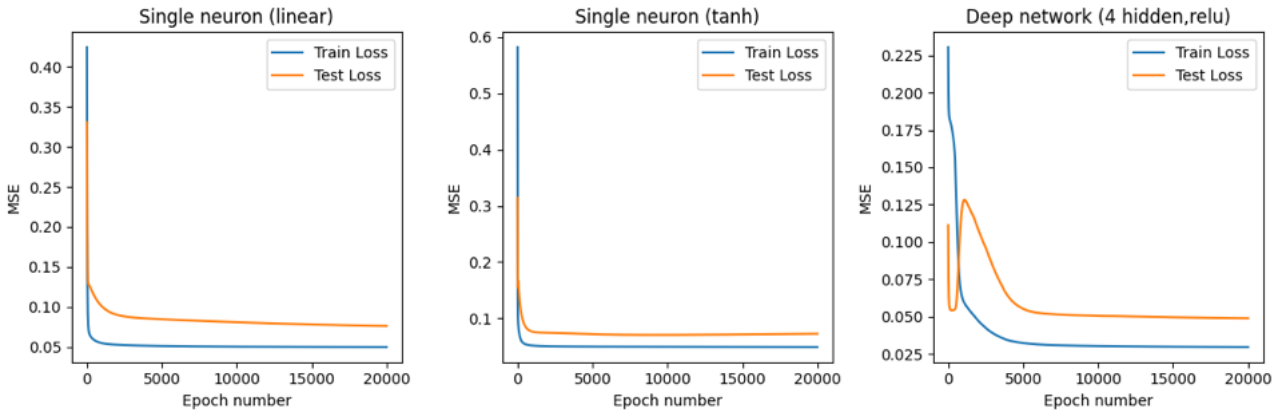


Figure 4.19: Smaller noise power=0.005

For the larger noise power, the DNN struggle to converge during training (instability) and the error during testing is significantly higher than those from the simple neurons. They are still able to show convergence albeit at a significantly higher error for both training and testing. For the DNN, the

poor result is due to over-fitting to the noise. For the lower noise power, all the models were able to converge since they are fitting more to the desired signals in stead of over-fitting to the noise. The DNN still maintains its position as the model with the least error for both training and testing in this case. Hence we can conclude that the DNN is effective when predicting highly non-linear data, but suffers greatly in performance when there is significant noise. The simple neurons are more robust the quality of data.

4.2 Interpretable NNS: Convolutional Neural Network for Frequency Decomposition

4.2.1 Sinusoidal Signal Generation

In Figure 4.20, the auto-convolution of signal y1, and the convolution of signals y1 with y2, and of y1 with y3, are plotted. The output amplitude of the auto-convolution is the largest. Due to the symmetry of the signals (cos functions), the convolution used may be interpreted as a cross-correlation, hence the greatest match happen with the same signal.

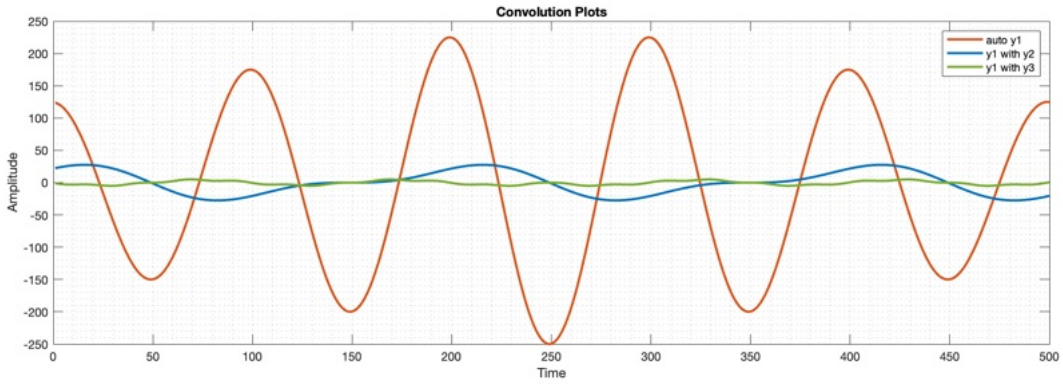


Figure 4.20: auto-conv of y1, and the conv of y1 with y2, and of y1 with y3,

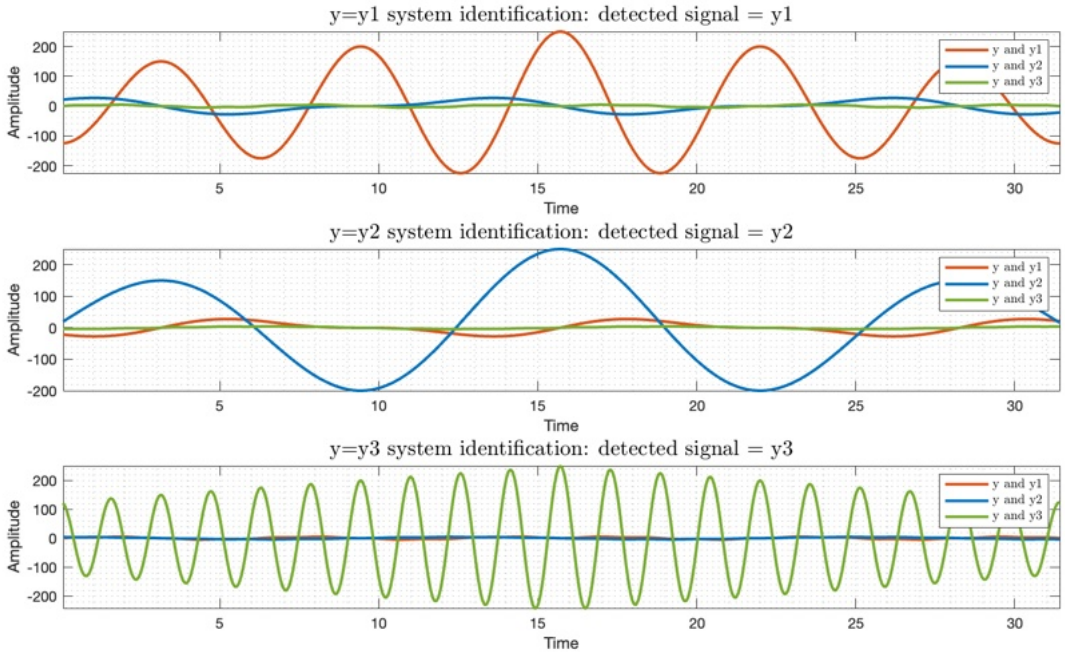


Figure 4.21: Classifying the signals

Only for this case, where the identification signals are symmetrical, convolution may be used. In when other cases, the signal of interest have to be flipped on its time axis before convolving it with the input test signal. After the convolution of the test signal with the three signals of interest, the maximum absolute value of the output is obtained and the correct classification of the signal would have the greatest value. This is shown in Figure 4.21.
In CNNs the kernel (or filter) is usually flipped before performing the dot product with the input tensor, where the notation reads $conv\{x, reverse(w)\}$. Because the mathematical expressions for convolution and the matched filter are:

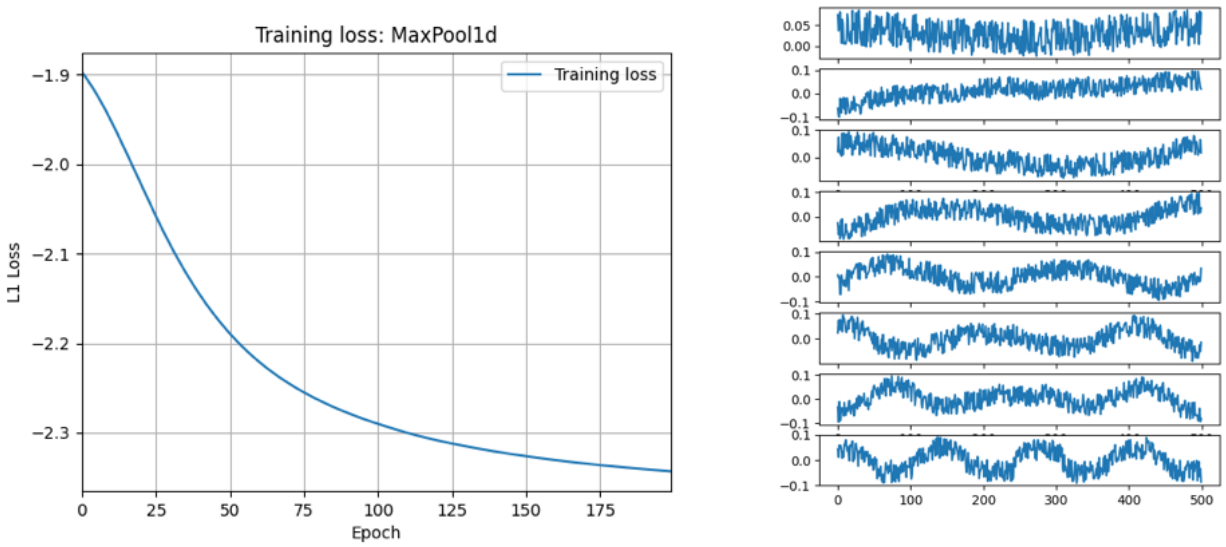
$$\textbf{Matched filter:} \qquad y_(n) = \sum_m x(m)w(n+m) \tag{89}$$

$$\textbf{Convolution:} \qquad y_(n) = \sum_m x(m)w(n-m) \tag{90}$$

The CNN is just simply implementing a cross correlation using the convolution operator.

4.2.2 Sinewave Frequency Classification Analysis

Based on the code snippet and default settings provided, the learning curve, a.k.a. the loss was plotted against epoch in Figure 4.22(a) and the converged kernels in Figure 4.22(b). Because the pooling effectively a form of downsampling used in neural networks, so the kernels represent different frequency components in the training signal where each weight is the amplitude of a signal of a certain frequency in the time window (kernel length = 500). You can make out the 8 known frequencies (0, $cos(0.5t)$, $cos(t)$,...,$cos(3.5t)$) that we used to generate the training data.



(a) Plot of log(L1loss)

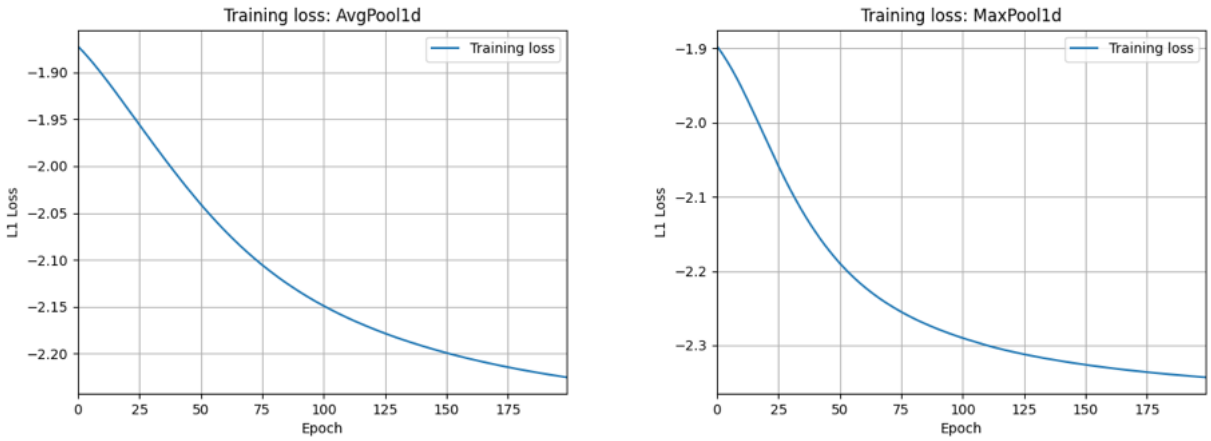
(b) Post-trained kernels

Figure 4.22: Learning curve and kernels from the default settings

Max pooling selects the maximum value from the pooled region. It tends to preserve the most dominant features and introduces sharp activations. It is generally more aggressive in selecting key features from the input, hence it typically results in faster convergence and a lower training loss in the short term. Average pooling takes the average of the values in the pooled region. This produces smoother activations and preserves more contextual information. It reduces overfitting and improve generalization, potentially leading to lower validation loss or a lower final training loss when regularization is important.

The model was run with AvgPool1d and MaxPool1d, then each of the L1loss over the 200 epochs

are presented in Figure 4.23. At closer inspection, by epoch 50, we can see that using Average Pooling, the model was able to converge to a loss of about $10^{-2.05}$ while using Max Pooling results in around $10^{-2.20}$. Hence in this case, Max Pooling provides a better performance due to the aggressive nature of selection.



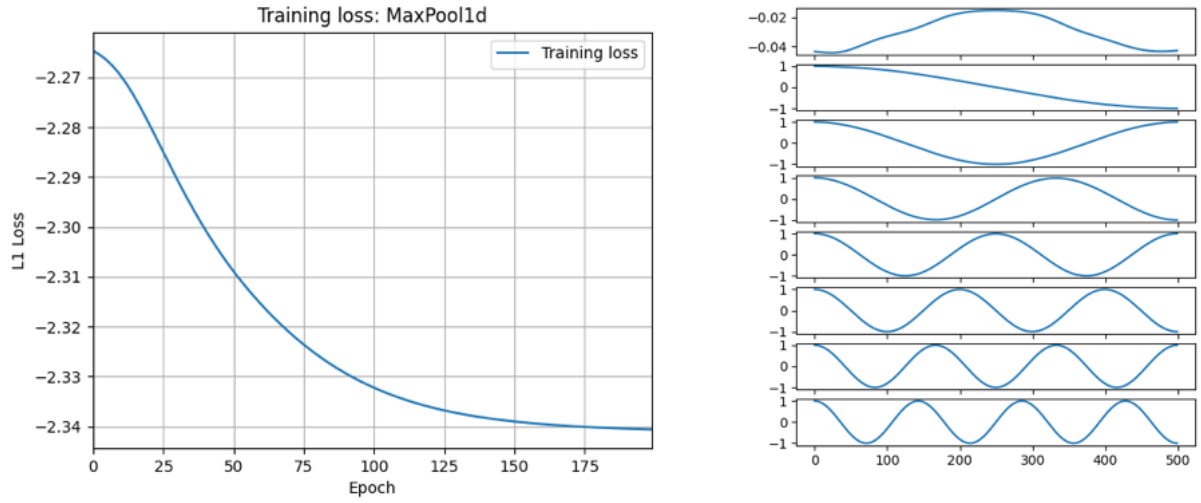
(a) Plot of log(L1loss) using Average Pooling

(b) Plot of log(L1loss) using Max Pooling

Figure 4.23: Loss against epoch, comparing the pooling methods

4.2.3 Kernel Initialization Impact on Convergence

The kernels are be initialised with the 8 known frequencies: 0, $cos(0.5t)$, $cos(t)$,...,$cos(3.5t)$, where there's a period of 2π every 500 samples. This should allow for a much faster convergence and a clearer kernel shape, making the training more computationally efficient and effective. Knowing what to **look for** is an easier task than finding patterns from scratch.



(a) Plot of log(L1loss)

(b) Post-trained kernels

Figure 4.24: Using kernels initialised with the 8 known signals and Average Pooling

The results are shown in Figure 4.24 and, as expected, the training starts at a lower loss and converges faster to an even lower error. The kernels maintain a clean 'shape' of the original initialisation, indicating that the prediction errors are low enough to not alter the weights significantly because the weights are close to ideal. It is the most likely contributor to the massive improvements in the training errors we observed. This showcases the use of domain knowledge to make these machine learning techniques more computationally efficient whilst producing stellar outputs.