Imperial College London

EE4-13 Adaptive Signal Processing and Machine Intelligence (2018-2019)

Imperial College London

Department of Electrical and Electronic Engineering

Coursework

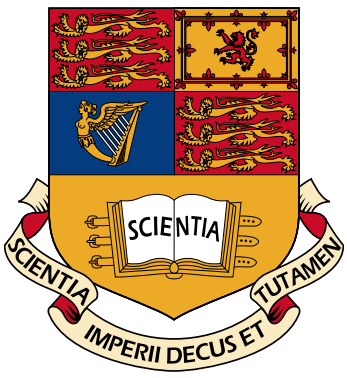
Authors: CID

Adel Haddad 01060023

Supervisor:

Prof. Danilo P. Mandic

Date: 12th April 2019



Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

1.1.a Approximation in the Definition of PSD

Starting with the equation provided, (1.1), we: use the relationship between modulus and complex conjugate for complex numbers, we move out the summation terms from the expectation operator, we factor out the exponential terms - as they are independent of the random variable x and we finally use the following substitution: $g(\tau) = r_{xx}(\tau)e^{-j\omega\tau}$.

$$
\begin{aligned}
P(\omega) &= \lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-j\omega n}\right|^2\right\} && (1.1)\\
&= \lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{N}\sum_{m=0}^{N-1} x(m)e^{-j\omega m}\sum_{k=0}^{N-1} x^*(k)e^{j\omega k}\right\}\\
&= \lim_{N\to\infty} \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} \mathbb{E}\left\{x(m)e^{-j\omega m}x^*(k)e^{j\omega k}\right\}\\
&= \lim_{N\to\infty} \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} \mathbb{E}\left\{x(m)x^*(k)\right\}e^{-j\omega(m-k)}\\
&= \lim_{N\to\infty} \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} r_{xx}(m-k)e^{-j\omega(m-k)} = \lim_{N\to\infty} \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} g(m-k) && (1.2)
\end{aligned}
$$

We can convert the double summation into a single summation using Equation 1.3, and recalling the earlier substitution:

$$\sum_{m=-N}^{N}\sum_{k=-N}^{N} g(m-k) = \sum_{\tau=-2N}^{2N}(2N+1-|\tau|)g(\tau) \tag{1.3}$$

(1.2) can then be written as:

$$
\begin{aligned}
P(\omega) &= \lim_{N\to\infty}\frac{1}{N}\sum_{\tau=-(N-1)}^{N-1}(N-|\tau|)r_{xx}(\tau)e^{-j\omega\tau}\\
&= \lim_{N\to\infty}\sum_{\tau=-(N-1)}^{N-1} r_{xx}(\tau)e^{-j\omega\tau} - \lim_{N\to\infty}\frac{1}{N}|\tau|\sum_{\tau=-(N-1)}^{N-1} r_{xx}(\tau)e^{-j\omega\tau}\\
&\approx \sum_{\tau=-\infty}^{\infty} r_{xx}(\tau)e^{-j\omega\tau} && (1.4)
\end{aligned}
$$

1.1.b Simulation of the Limiting Case

An unbiased estimator of the autocorrelation will provide a case where the correlation does not rapidly decay. This would violate the derivation shown in Section 1.1.a, thereby hindering equivalence. An example is shown in Figure 1.1.1, here we see a sine function with an unbiased estimator:

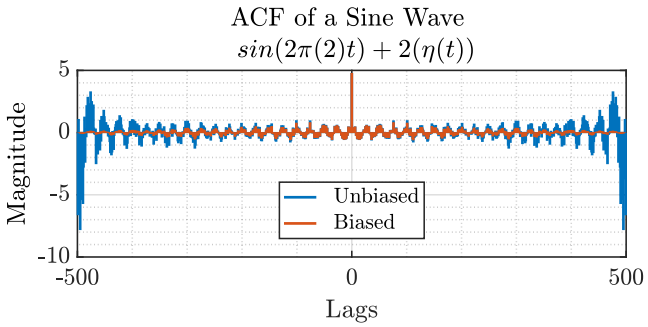


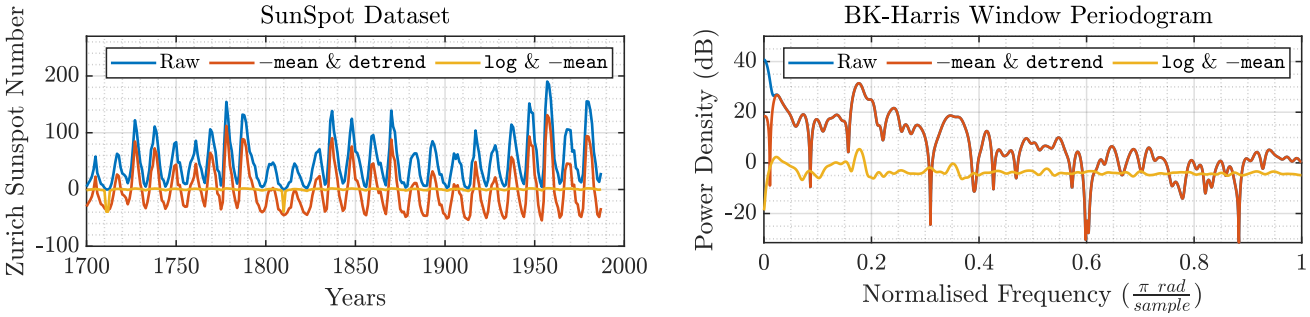
Figure 1.1.1: An example of the Limiting Case of the Periodogram Definition

1.2 Periodogram-based Methods Applied to Real–World Data

1.2.a The SunSpot Dataset

Figure 1.2.1. The mean's influence on data is the offset DC bias, captured in the $f = 0$ component of the periodogram. Hence as we would expect, subtracting the `mean` reduces its magnitude in the periodogram. `detrend` removes linear trends, it seems in the case of this data set that most linear trends are captured at $f \lessapprox 0.02 rad/sample$.

The natural logarithm was taken using: `log`. As the logarithm has a compression effect on magnitude we see that the magnitude of both raw and periodogram is greatly attenuated. We note that frequencies of interest and its harmonics appear as more distinct when compared to the rest of the periodogram.

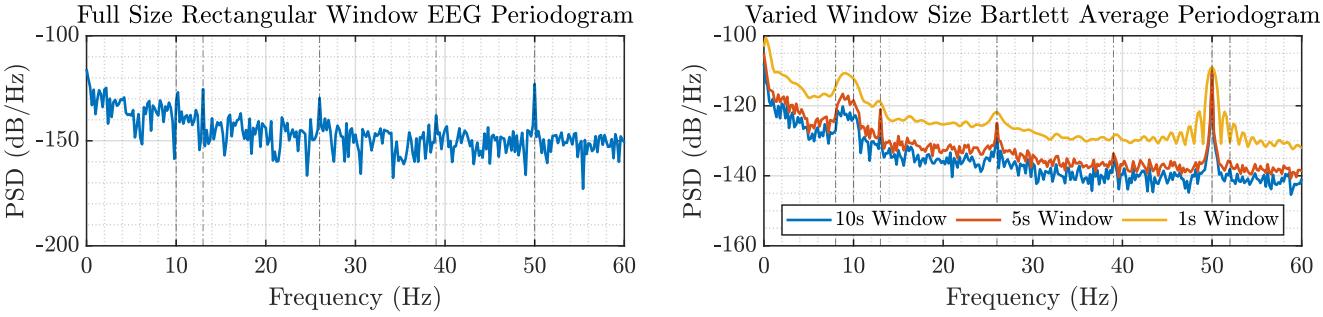


(a) Raw and its preprocessed datas

(b) Periodograms

Figure 1.2.1

1.2.b The EEG Dataset



(a) Standard Periodogram

(b) Bartlett Average Periodograms

Figure 1.2.2

Response	Expected Range (Hz)	Observed Range (Hz)
Alpha Rhythm	$8-10$	$8-10$
SSVEP	$\texttt{range}[11-20]$	$13n$
Power-Line	50	50

Table 1.2.1: EEG Frequency Peaks of the Periodogram. n refers to harmonics

The standard periodogram has identifiable peaks, showing in Figure 1.2.2, quantified in Table 1.2.1. We are unable to identify the 3rd harmonic of the SSVEP, at 52Hz it is too close to the power-line interference at 50Hz. The main difference in the 10s window averaged periodogram is clearer peak isolation compared to the surrounding periodogram and emphasis on the range of frequencies of the alpha-rhythm, rather than a single discrete peak.

1.3 Correlation Estimation

1.3.a Unbiased and Biased ACF Estimates

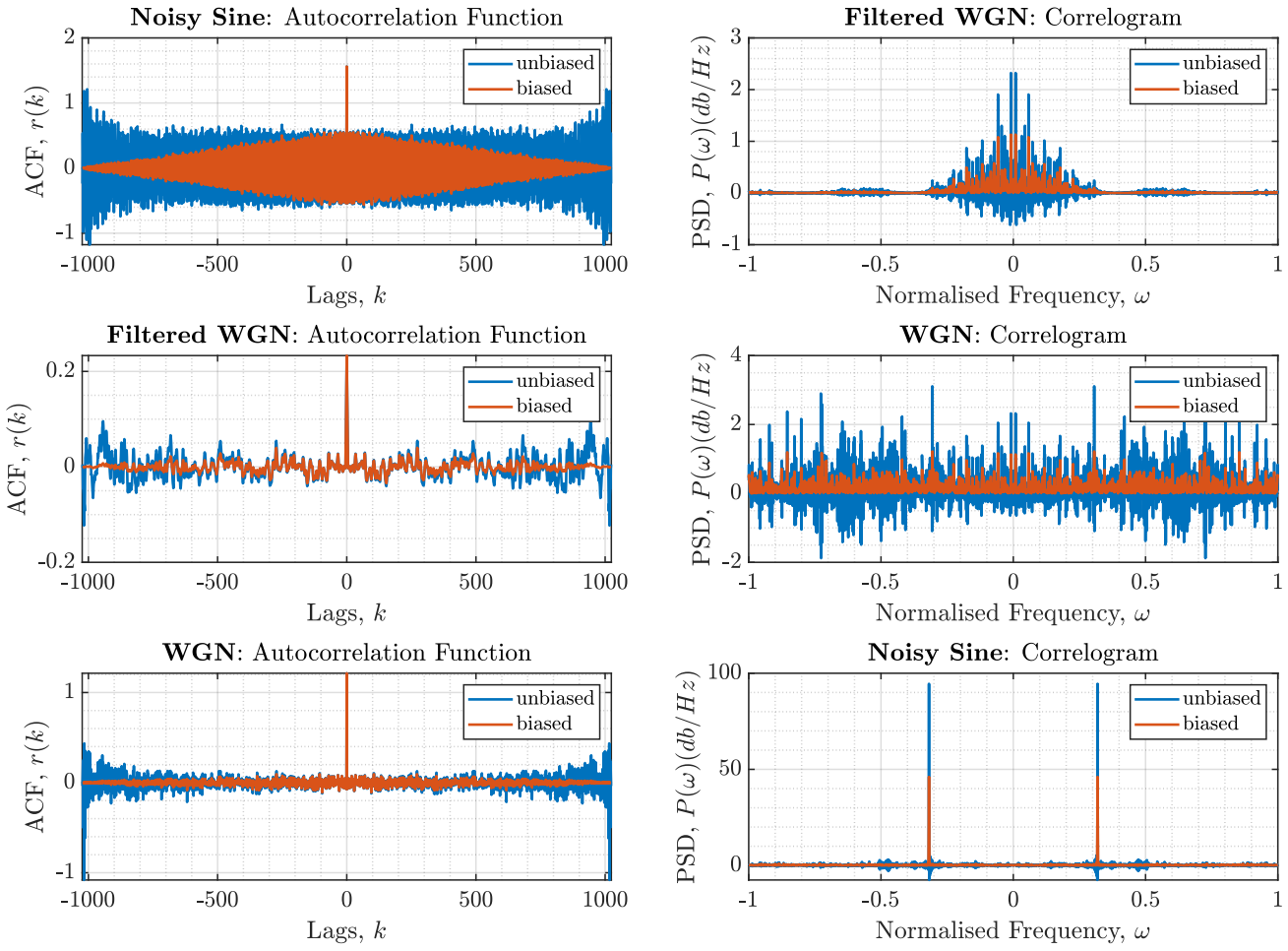


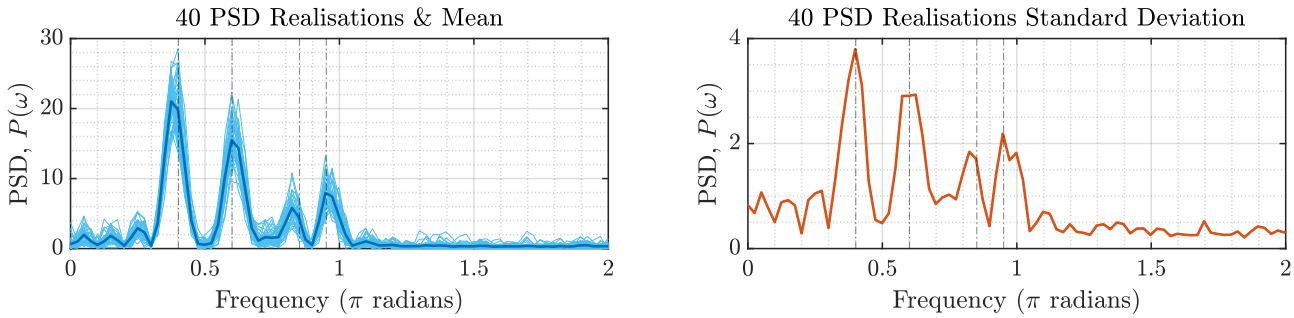
Figure 1.3.1: Set of Auto-Correlation Functions (ACFs) and their Correlograms

Figure 1.3.1. For the autocorrelation functions: we can see that the biased estimator tends to 0 for increasing lag magnitude, whereas the unbiased estimator remains somewhat constant, although at the extremes it begins to increase to approximately double the constant value.
For the correlograms: we observe that the biased estimator does not contain negative values and that the unbiased estimator is approximately similar for smaller lags.

1.3.b Biased ACF Estimator PSDs

The process simulated was the following:

$$\begin{aligned} x(n) =& 2sin(2\pi 0.4n) + 1.75sin(2\pi 0.6n) \\ &+ 0.85sin(2\pi 0.85n) + 1.2sin(2\pi 0.95n) + \eta(n) \quad \eta \sim \mathcal{N}(0,1) \end{aligned} \tag{1.5}$$

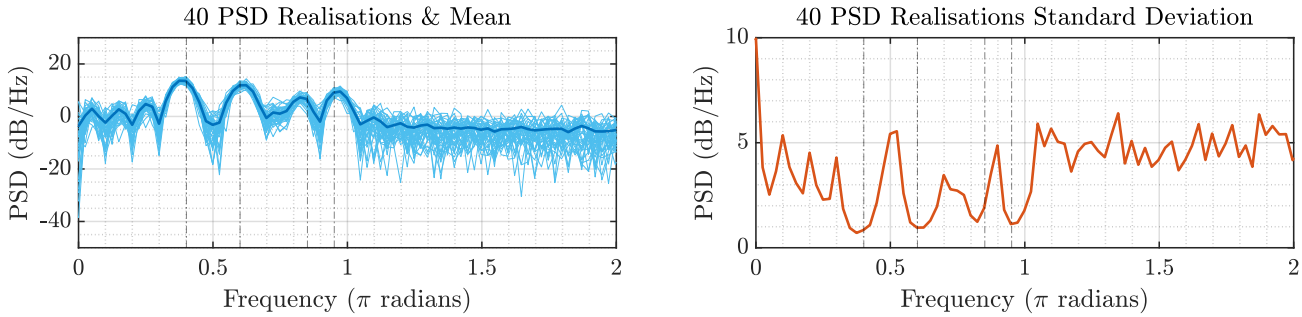


(a) Periodogram (b) Standard Deviation

Figure 1.3.2: The total number of data points used was 512, black vertical lines indicate the model defined frequencies

It is interesting to see the low frequency resolution influences the accuracy of the peak with respect to the actual frequencies used, Figure 1.3.2.

1.3.c Biased ACF Estimator PSDs on the dB Scale



(a) Periodogram (b) Standard Deviation

Figure 1.3.3: The total number of data points used was 512, black vertical lines indicate the model defined frequencies

Figure 1.3.3. It is advantageous that the standard deviation decreases around our frequencies as be seen at the 4th frequency, where on Figure 1.3.2, the peak was ambiguous, here as a trough it is much better defined.

1.3.d Influence of Data Samples on the PSD

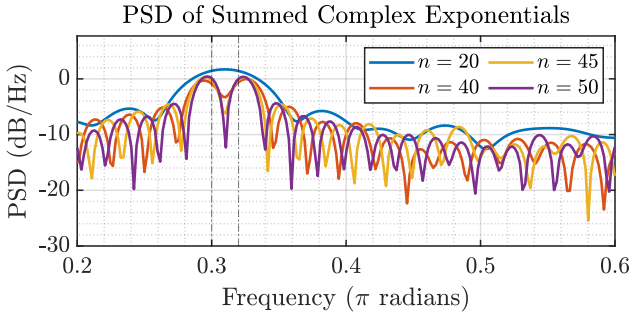


Figure 1.3.4: PSD while varying n, the number of Data Samples used

In Figure 1.3.4 we can clearly see that the frequency resolution is insufficient at lower sample numbers, resulting in aliasing of the desired frequency peaks.

1.3.e MUltiple SIgnal Classification (MUSIC) Estimator

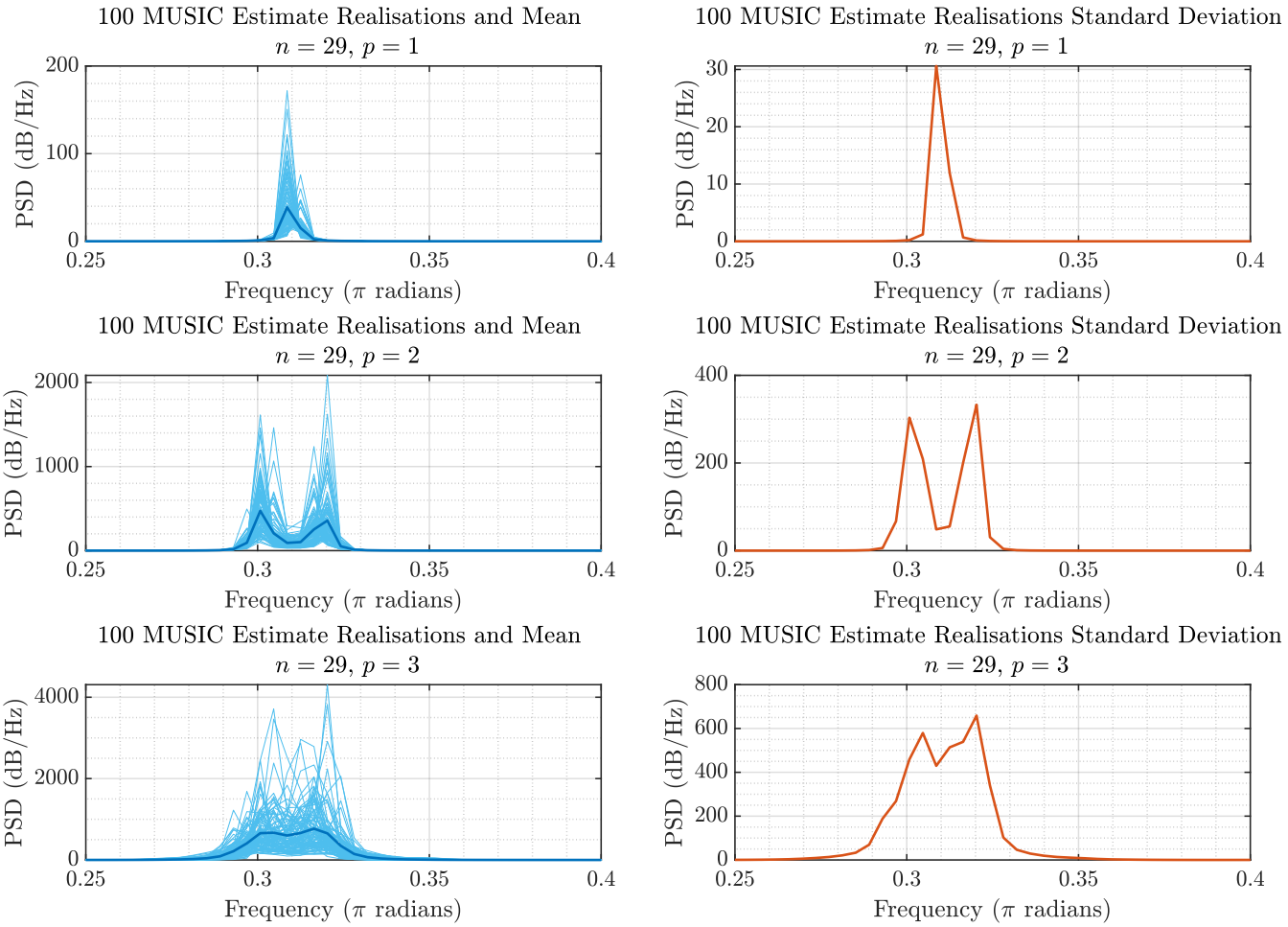


Figure 1.3.5: Set of Auto-Correlation Functions (ACFs) and their Correlograms.
n is the number of samples used, p is the Signal Space Dimensionality

The MUSIC estimate identifies eigenvalues of an autocorrelation matrix in descending magnitude order - which indicate directions of largest variability in the subspace.
The first line creates the 14 dimensional modified correlation matrix R_{xx} for the dataset, which is

subsequently used by the MUSIC algorithm with p setting the Signal Subspace Dimensionality in the second line.
The third line plots the Pseudospectrum (PSD Estimate) against the frequency axis.

Figure 1.3.5 shows the MUSIC estimate plots for the given equation. The spectrum is not very detailed, peaks are not all sharp, but are clearly distinguishable from the surrounding periodogram. In the general case the MUSIC estimator is a good choice if the signal space dimensionality is known, especially as it works on such a limited set of samples.

The periodogram is equally suitable for resolving peaks, but requires more samples for a meaningful resolution.

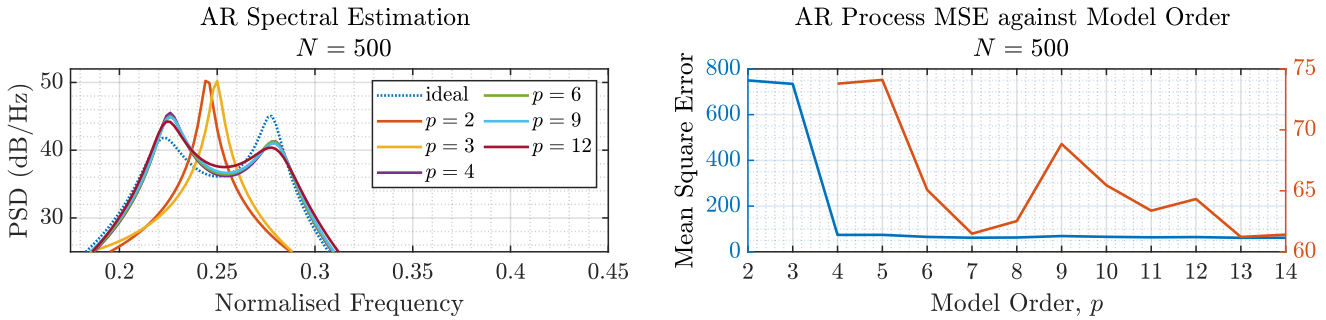
In both periodogram and MUSIC estimators we see that the standard deviation, hence variance increase around the peaks, except when using the log scale of the periodogram, where it decreases.

1.4 Spectrum of Autoregressive (AR) Processes

1.4.a Shortcomings of the Unbiased ACF in finding AR Parameters

As the unbiased estimator allows for negative values, at a computational level it will require more bits to store, especially for larger values.

1.4.b Error of the AR PSD Estimate



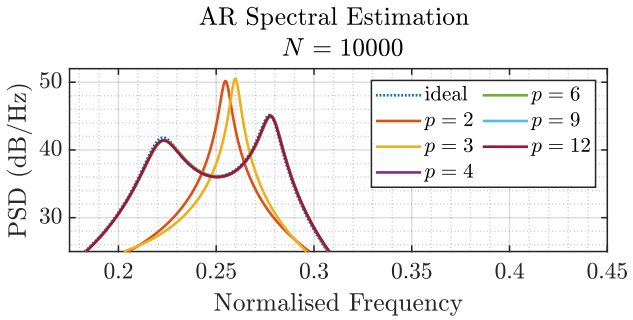
(a) AR Periodogram and its p Order Estimates

(b) Mean Squared Error with two y-scales

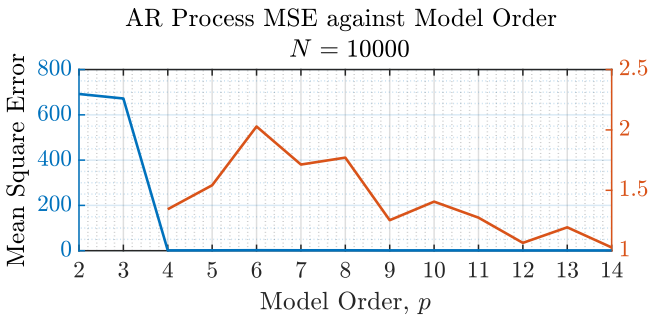
Figure 1.4.1

We can see in (a) of 1.4.1, that increasing the order tends towards a better solution. But (b) notes that the most drastic difference is at the model order, 4, which matches the order of the process defined, subsequent increases of the model order do increase its likeliness to the true response, but changes are not so drastic.

1.4.c Error of the AR PSD Estimate with more Samples



(a) AR Periodogram and its p Order Estimates



(b) Mean Squared Error with two y-scales

Figure 1.4.2

The same trend is seen in Figure 1.4.2, as we saw with $N = 500$, although the estimate now matches the underlying model much better, reflected in the drastically lower Mean Square Error (MSE).

It is noted that for a more valid comparison of model order's influence on the estimate's error the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) are more suitable quantifiers of model order suitability than MSE. But the accuracy trend reflected in the MSE is still valid for our comparisons.

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

1.5.a Standard & Average PSDs of the RRI Dataset

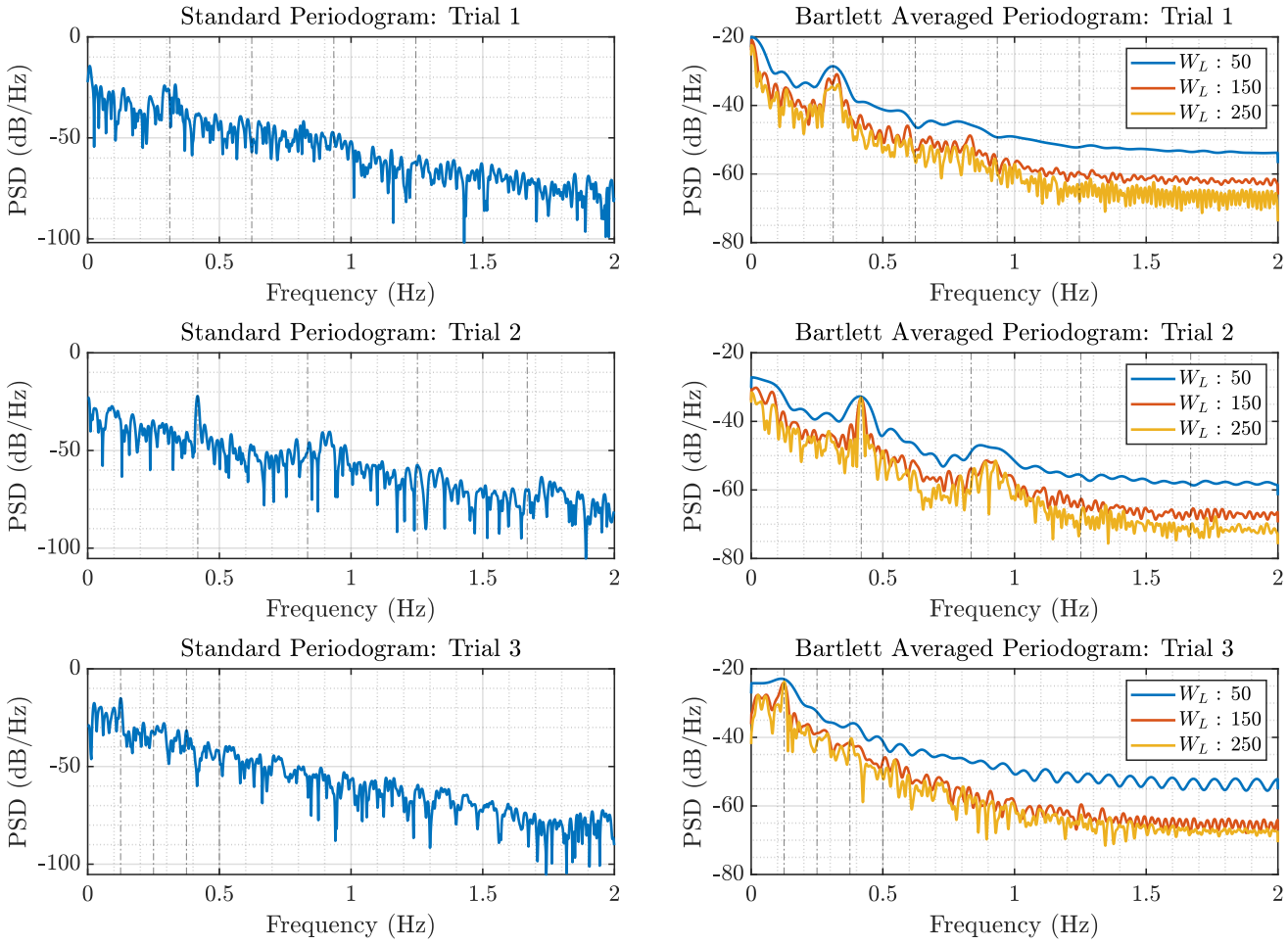


Figure 1.5.1: Standard and Bartlett Average Periodograms.
W_L is the Window Length used. First 4 harmonics denoted by vertical black lines. Zero Padded Signal Length: 4096.

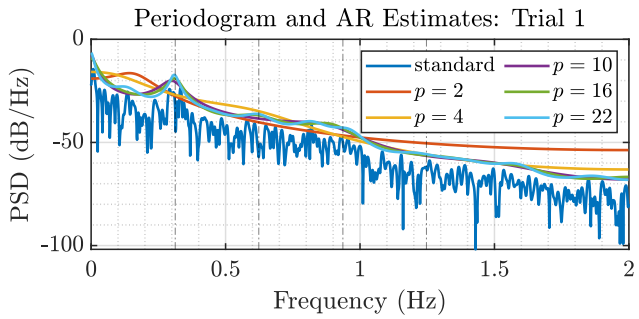
Breathing Type	**Expected** (BPM)	**Observed Peak** (BPM)
Normal (Trial 1)	$10 - 15$	$0.31 \times 60 \approx 18.7$
Fast (Trial 2)	25	$0.41 \times 60 \approx 25$
Slow (Trial 3)	7.5	$0.125 \times 60 = 7.5$

Table 1.5.1: Breaths Per Minute (BPM), i.e. Observed Frequency $\times 60$, for all trials.

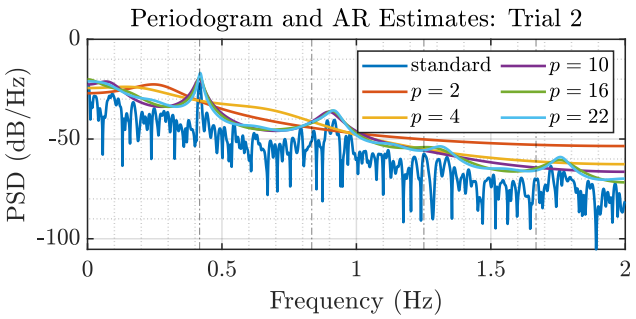
1.5.b Analysis of the RRI PSD Estimates

Figure 1.5.1 shows us distinct peaks for each trial that somewhat agrees with the breathing expected. The performance of the $W_L = 50$ is also sufficiently accurate for Trials 1 and 2, however less so for Trial 3. We note that harmonics are difficult to distinguish, despite the large zero padding of the signal. Table 1.5.1 highlights how the observed peaks align with the expected peaks.

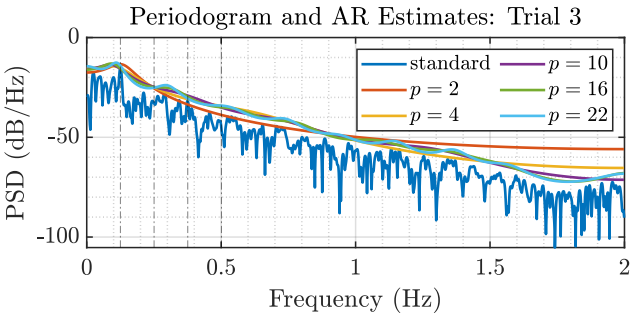
1.5.c AR PSD Estimate for the RRI Dataset



(a)



(b)



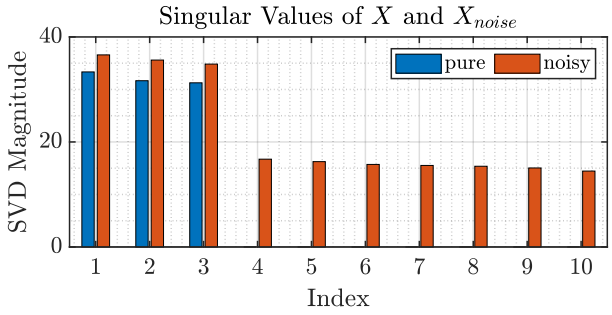
(c)

Figure 1.5.3: AR Estimate Periodograms. p is the model order. Black lines are frequency of interest and its harmonics

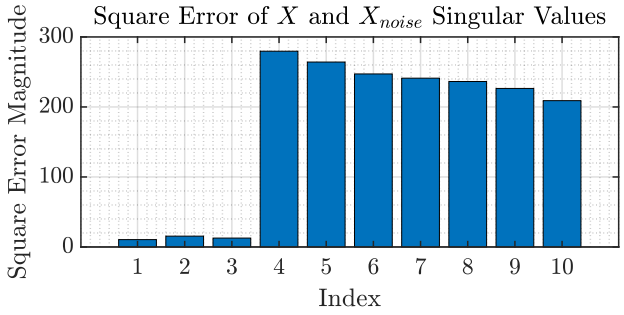
The AR spectral estimate in Figure 1.5.3 correctly identified the peak from models of order $p \gtrapprox 10$. We note that the estimate resembles a smooth envelope over our PSD and correctly identifies clear peaks in the standard PSD at $p = 10$. We also can see for higher order models that there are move fluctuations as the model starts to overfit for the input data's natural noisiness. Compared to our periodogram estimate we are better able to find harmonics in Trial 2 with the AR model, but unfortunately they are not accurate.

1.6 Robust Regression

1.6.a Single Value Decomposition (SVD)



(a) SVD



(b) Square Error

Figure 1.6.1

The rank of the input data would be indicated when the SVD Error increases or the non-zero SVD values end (for pure data) - hence the data is of rank 3.
Noise makes the SVD magnitude non-zero where we expect it to be zero for the pure input. Hence, in the real world where we would not have the pure dataset we would need to purely look at Figure 1.6.1 (a), if the SVD magnitude of the noise was comparable to that of the signal we would have difficulty establishing a criteria to distinguish signal and noise subspaces from the SVD method.

1.6.b Low Rank Approximation Error

By looking at the mean square error between the pure signal and the low-rank approximation of the noise, Figure 1.6.2, we note that the minimum error is when the low-rank approximation matches that of the pure data. This clear difference would allow automated algorithms to search for the minimum MSE error and correct obtain the appropriate model order.

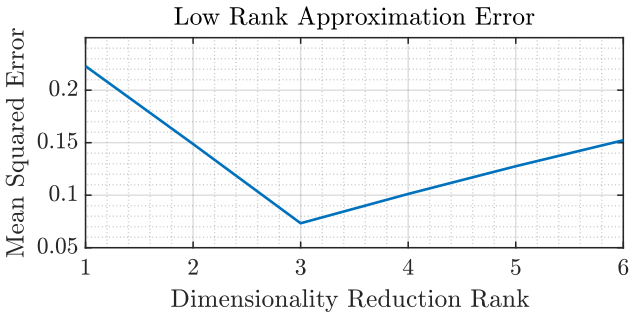
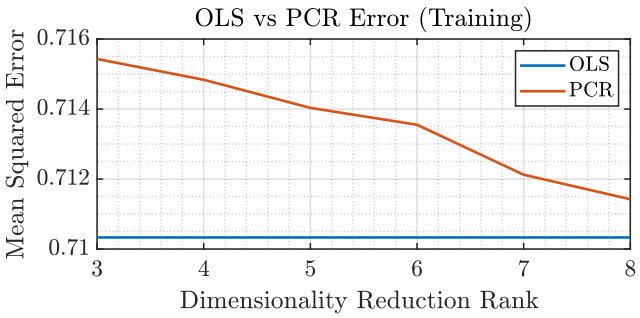
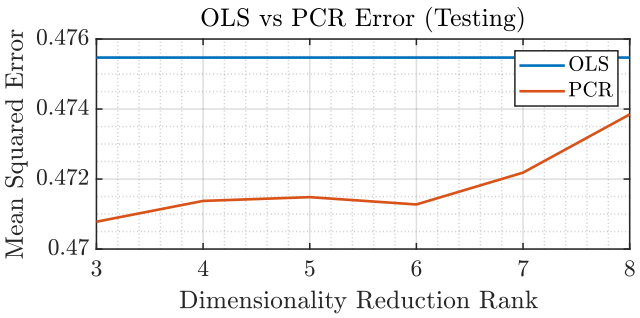


Figure 1.6.2: Effect of Changing Rank on the Approximation Error

1.6.c Ordinary Least Squares (OLS) & Principle Component Regression (PCR) Estimate Errors



(a) Training Dataset Error



(b) Testing Dataset Error

Figure 1.6.3: Comparison of Errors in the Training and Testing Datasets

In both datasets we see that at the true model order the differences are tiny, in the training dataset we see that PCR underperforms by 0.85%, while in the testing dataset PCR outperforms by 1.05%. We can also note that decreasing the PCR model order further than the true model order, improves performance in the Training dataset but reduces performance in the Testing dataset.

1.6.d Ordinary Least Squares (OLS) & Principle Component Regression (PCR) Estimate Errors - Part 2

Again if we look at the true model order, Figure 1.6.4, PCR outperforms OLS by 0.79%. We note how further reduction in dimensionality decreases the performance of PCR. In regard to effectiveness of each scheme, if the model order is unknown and cannot be approximated accurately, OLS is a safer choice, however best performance - although marginal - can be achieved if the model order is known and PCR is implemented.

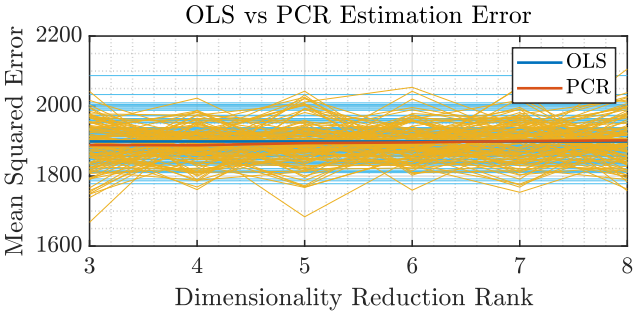


Figure 1.6.4: Mean Square Error over several Realisations

2 Adaptive Signal Processing

2.1 The Least Mean Square (LMS) Algorithm

2.1.a Example Correlation Matrix

An AR(2) process $x(n)$ with parameters a_1, a_2 satisfies the difference equation:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n) \tag{2.1}$$

where $\eta(n) \sim \mathcal{N}(0, \sigma_\eta^2)$. The Least Mean Square (LMS) algorithm is used to approximate the autoregressive parameters a_1, a_2 from data, with input $\mathbf{x}(n) = [x(n-1), x(n-2)]^T$ and output $y(n) = x(n)$.

We will first check for stationarity, we know that for an AR(2) process to be stationary the roots of its characteristic polynomial must lie outside the unit circle, which can be confirmed if the following are satisfied:

$$a_1 + a_2 < 1 \tag{2.2}$$

$$a_2 - a_1 < 1 \tag{2.3}$$

$$|a_2| < 1 \tag{2.4}$$

We can confirm is valid for our AR(2) process, which has $a_1 = 0.1$, $a_2 = 0.8$.
The correlation matrix $\mathbf{R}_{xx}$ of the stationary input is thereby given by:

$$\mathbf{R}_{xx} = \mathbb{E}[\mathbf{x}(n)\mathbf{x}^T(n)] = \mathbb{E}\begin{bmatrix} x(n-1)x(n-1) & x(n-1)x(n-2) \\ x(n-2)x(n-1) & x(n-2)x(n-2) \end{bmatrix} = \begin{bmatrix} r_{xx}(0) & r_{xx}(1) \\ r_{xx}(1) & r_{xx}(0) \end{bmatrix} \tag{2.5}$$

where $r_{xx}(k)$ is the autocorrelation function (ACF) of $x(n)$. To obtain the ACF of the AR(2) process, we must then multiply equation (2.1) by $x(n-k)$ and take the Expectation, $\mathbb{E}[\cdot]$:

$$\begin{aligned} r_{xx}(k) &= \mathbb{E}[x(n)x(n-k)] \\ &= \mathbb{E}\left[a_1 x(n-1)x(n-k) + a_2 x(n-2)x(n-k) + \eta(n)x(n-k)\right] \\ &= a_1 \mathbb{E}\left[x(n-1)x(n-k)\right] + a_2 \mathbb{E}\left[x(n-2)x(n-k)\right] + \mathbb{E}\left[\eta(n)x(n-k)\right] \end{aligned} \tag{2.6}$$

$\mathbb{E}[\eta(n)x(n-k)]$ goes to 0 when $k > 0$:

$$r_{xx}(0) = a_1 r_{xx}(1) + a_2 r_{xx}(2) + \sigma_\eta^2, \qquad k = 0 \tag{2.7}$$

$$r_{xx}(k) = a_1 r_{xx}(k-1) + a_2 r_{xx}(k-2), \qquad k > 0 \tag{2.8}$$

Using the true process parameters $a_1 = 0.1$, $a_2 = 0.8$ and $\sigma_\eta^2 = 0.25$, together with the even property of the ACF, $r_{xx}(-k) = r_{xx}(k), \forall k \in \mathbb{N}$, we obtain three simultaneous equations with three unknowns:

$$r_{xx}(0) = a_1 r_{xx}(1) + a_2 r_{xx}(2) + \sigma_\eta^2 \tag{2.9}$$

$$r_{xx}(1) = a_1 r_{xx}(0) + a_2 r_{xx}(-1) = a_1 r_{xx}(0) + a_2 r_{xx}(1) \tag{2.10}$$

$$r_{xx}(2) = a_1 r_{xx}(1) + a_2 r_{xx}(0) \tag{2.11}$$

with the unique solution $r_{xx} = [\frac{25}{27}, \frac{25}{54}, \frac{85}{108}]$ for $k = 0, 1, 2$.
Hence by substitution in equation (2.5) we obtain the correlation matrix of the input vector $\mathbf{x}(n)$:

$$\mathbf{R}_{xx} = \frac{25}{54}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{2.12}$$

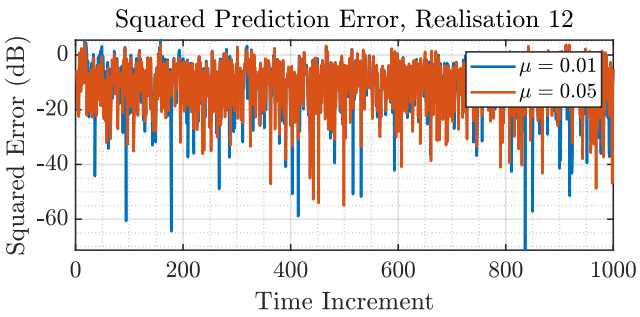
Convergence of the LMS algorithm depends on the step size μ, which should satisfy:

$$0 < \mu < \frac{2}{\lambda_{max}} \tag{2.13}$$

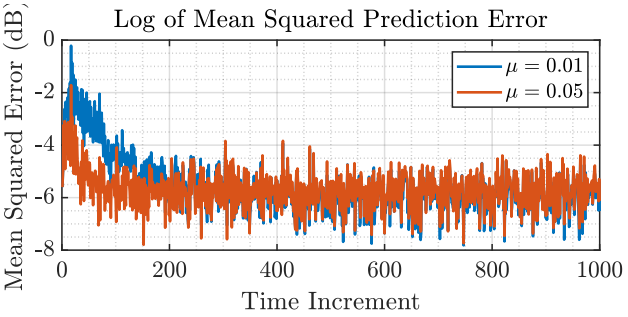
where λ_{max} the largest eigenvalue of the correlation matrix $\mathbf{R}_{xx}$. Performing the eigendecomposition of $\mathbf{R}_{xx}$, we obtain $\lambda_1 = 0.4630$ and $\lambda_2 = \lambda_{max} = 1.3889$ and as a result, convergence to the Wiener optimal solution is guaranteed for:

$$0 < \mu < 1.44 \tag{2.14}$$

2.1.b Example Learning Curve for an AR(2) Process



(a) Square Prediction Error, Single Realisation



(b) Log of Mean of the Squared Prediction Error

Figure 2.1.1: μ's effect on the Squared Prediction Error and the Influence of the Log operator

As can be seen in Figure 2.1.1, the log operator exponentially emphasises the small trends in the error signal. It is now easier to see the influence of the higher μ on reaching steady state, where we can see that larger values converge faster.

2.1.c Misadjustment

The theoretical Misadjustment, $\mathcal{M}_{LMS}$, of the LMS algorithm can be approximated with:

$$\mathcal{M}_{LMS} \approx \frac{\mu}{2} Tr\{\mathbf{R}_{xx}\} = \frac{\mu}{2}\left(\frac{100}{54}\right) = \frac{\mu}{2} 1.8519 \tag{2.15}$$

where $\mathbf{R}_{xx}$ from (2.12) is used.

Figure 2.1.1 shows that the squared prediction error converges for both μ after $t > 200$. Looking at $t > 400$ to guarantee that a steady-state has been reached, then a Mean Squared Error (MSE) can be estimated. The empirical misadjustment, $\mathcal{M}_{emp}$, is found with:

$$\mathcal{M}_{emp} = \frac{\texttt{EMSE}}{\sigma_\eta^2} = \frac{\texttt{MSE}}{\sigma_\eta^2} - 1 \tag{2.16}$$

Table 2.1.1 summarises the empirical, theoretical and practical misadjustments of the simple LMS algorithm for the $x(n)$ process.

μ	$\mathcal{M}_{emp}$	Theoretical $\mathcal{M}_{LMS}$	Practical $\mathcal{M}_{LMS}$
0.01	0.0093	0.0077	0.0182 ± 0.05619
0.005	0.0463	0.0491	0.0593 ± 0.06409

Table 2.1.1: Theoretical and LMS Estimated Misadjustments

Misadjustment for larger μ values appears to be larger, corresponding to what is observed in Figure 2.1.1 and 2.1.3, where larger MSE is contributing to this difference. It is clear that the standard deviations of misadjustment are very large in the practical data.

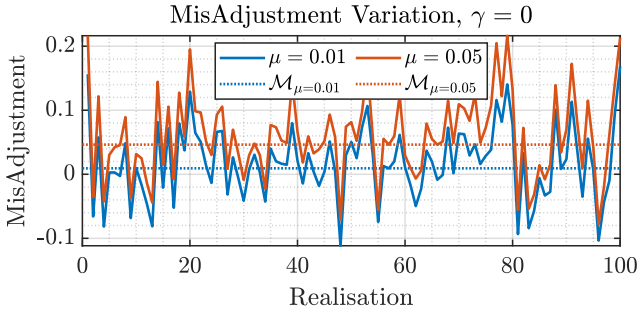


Figure 2.1.2: Estimated Misadjustment, varying with realisation. Theoretical $\mathcal{M}_{LMS}$ is additionally plotted for reference.

2.1.d Steady State Estimation

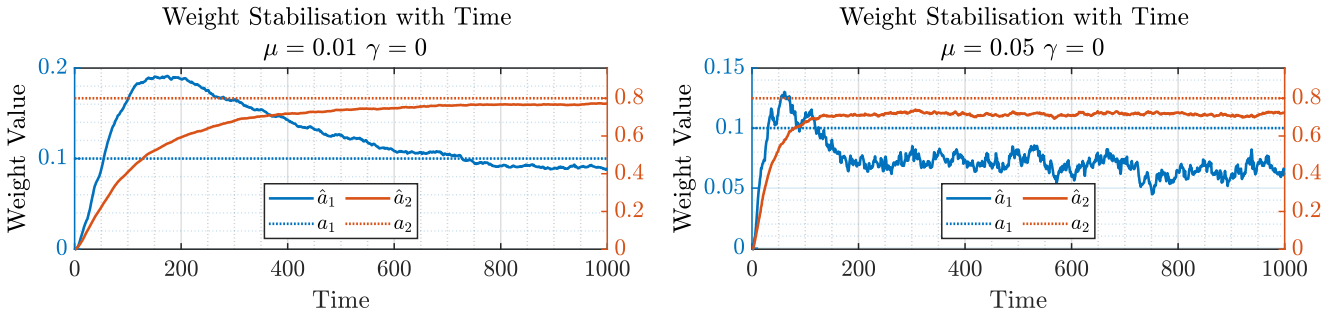


Figure 2.1.3: μ's effect on weight stabilisation

The simulation shown in Figure 2.1.3, shows that convergence to a steady state is faster with larger μ values, 200 ($\mu = 0.05$) vs 900 ($\mu = 0.01$), however, we clearly observe the tradeoff where the converged value is less accurate and more noisy, as can be clearly seen for $\hat{a}_1$.

2.1.e Leaky LMS Derivation

The cost function $\mathcal{J}_2$:

$$\mathcal{J}_2(n) = \frac{1}{2}\Big(e^2(n) + \gamma\|\boldsymbol{w}(n)\|_2^2\Big) \tag{2.17}$$

Expressing the error term, $e(n)$, in terms of the input vector $\mathbf{x}_n$ and the (generic) target y_n:

$$\mathcal{J}_2(n) = \frac{1}{2}\Big(\|y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n)\|_2^2 + \gamma\|\boldsymbol{w}(n)\|_2^2\Big) \tag{2.18}$$

$$= \frac{1}{2}\Big(\big(y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n)\big)^T\big(y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n)\big) + \gamma\boldsymbol{w}(n)^T\boldsymbol{w}(n)\Big) \tag{2.19}$$

Calculating the gradient of the cost function, $\nabla_{\boldsymbol{w}}\mathcal{J}_2$, with respect to the weights $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}}\mathcal{J}_2(n) = \frac{1}{2}\frac{\partial}{\partial \boldsymbol{w}}\Big(\big(y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n)\big)^T\big(y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n)\big) + \gamma\boldsymbol{w}(n)^T\boldsymbol{w}(n)\Big) \tag{2.20}$$

$$= \frac{1}{2}\big(-2(y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n))\mathbf{x}(n) + 2\gamma\boldsymbol{w}(n)\big) \tag{2.21}$$

$$= -(y(n) - \boldsymbol{w}(n)^T\mathbf{x}(n))\mathbf{x}(n) + \gamma\boldsymbol{w}(n) \tag{2.22}$$

$$= -e(n)\mathbf{x}(n) + \gamma\boldsymbol{w}(n) \tag{2.23}$$

We take the negative of this gradient, i.e. gradient descent, hence the weight update equation becomes:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu\nabla_{\boldsymbol{w}}\mathcal{J}_2(n) \tag{2.24}$$

$$= \boldsymbol{w}(n) - \mu\big(-e(n)\mathbf{x}(n) + \gamma\boldsymbol{w}(n)\big) \tag{2.25}$$

$$= (1 - \mu\gamma)\boldsymbol{w}(n) + \mu e(n)\mathbf{x}(n) \tag{2.26}$$

Hence we proved that the Leaky LMS algorithm weight update rule, (2.26), is derived from the minimisation of the cost function $\mathcal{J}_2$, (2.17).

2.1.f Example Leaky LMS for an AR(2) Process

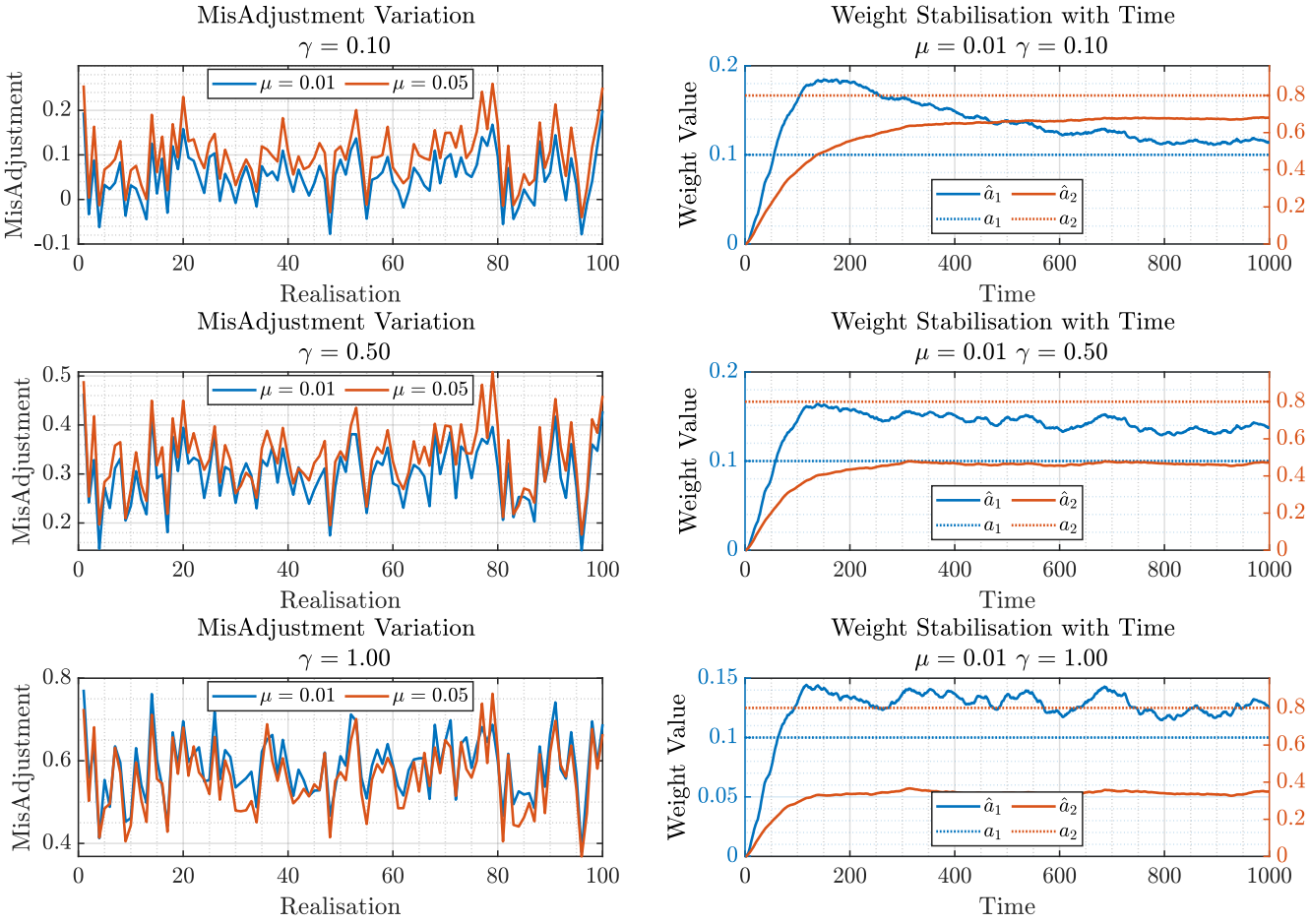


Figure 2.1.4: γ's leakage, forgetful, effect on weight stabilisation

The leaky LMS implements the term γ, which we can interpret as a forgetfulness term, where the LMS filter forgets the previous weight's significance. This means that the filter reaches a steady state faster, which can be observed in Figure 2.1.4, where we see convergence is 400 steps instead of 900 when using $\gamma = 0.5$.
Importantly, the accuracy is greatly compromised, the magnitude of the error is directly proportional to the γ used.
The usefulness of the leaky LMS is more apparent when the μ value is unstable in the original LMS implementation, now the leaky LMS is capable of convergence whereas the original LMS will not converge to a reasonable value.

2.2 Adaptive Step Sizes

2.2.a Gradient Adaptive Step Sizes (GASS) Comparisons for a MA(1) System

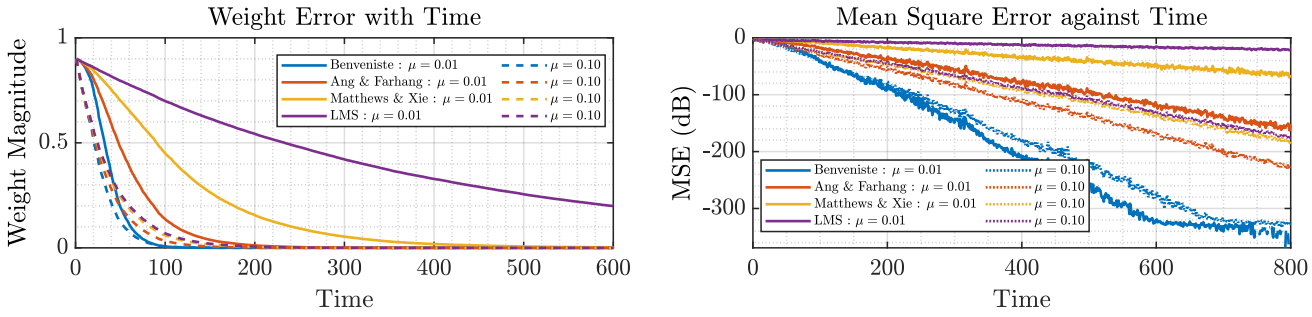


Figure 2.2.1: Weight Error and Prediction Error for various GASS algorithms

In brief, the step size is modulated in: Matthews & Xie utilises by the gradient of the error, Ang & Farhang by a static low pass to reduce noise effects and in Benveniste by an adaptive low pass. Varying our μ term would allow us to learn faster, by taking larger steps of change in our weights, for example when the error is large. What we observe in the gradients in the (log emphasised) mean square error plot, Figure 2.2.1, is the effective learning rate for these algorithms. Additionally we can see the influence of changing the algorithm's starting learning rate, and its effect on convergence. It is clear that the Beneviste algorithm has the faster convergence, but when looking at the effect of increasing μ we observe the improvement of learning rate (increase of gradient magnitude) in descending order is: LMS, Matthews & Xie then Ang & Farhang.
In the highly dynamic applications where such filters would be needed, we can imagine that the learning rate can influence the output properties when compared to the desired output. For example, if the learning rate was less than or equal to the applications fluctuation rate we would observe a low pass filter effect, as the predicted output would not necessarily reach the actual output in the time between fluctuations.

2.2.b NLMS Update Equation Equivalence

Starting from the update equation based on the *a posteriori error* $e_p(n) = d(n) - \mathbf{x}(n)^T\boldsymbol{w}(n+1)$:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e_p(n)\mathbf{x}(n) \tag{2.27}$$

Multiply both sides with $-\mathbf{x}(n)^T$ and add $d(n)$ to create $e_p(n)$:

$$d(n) - \mathbf{x}(n)^T\boldsymbol{w}(n+1) = d(n) - \mathbf{x}(n)^T\boldsymbol{w}(n) - \mathbf{x}(n)^T\mu e_p(n)\mathbf{x}(n) \tag{2.28}$$

The LHS term is the *a posteriori* error, $e_p(n)$, while the first RHS term the *a priori error*, $e(n)$:

$$e_p(n) = e(n) - \mu e_p(n)\|\mathbf{x}(n)\|^2 \tag{2.29}$$

$$e_p(n) = e(n)\frac{1}{1+\mu\|\mathbf{x}(n)\|^2} \tag{2.30}$$

Substituting (2.30) into the weight update equation:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\frac{1}{1+\mu\|\mathbf{x}(n)\|^2}e(n)\mathbf{x}(n) \tag{2.31}$$

$$= \boldsymbol{w}(n) + \frac{1}{\frac{1}{\mu}+\|\mathbf{x}(n)\|^2}e(n)\mathbf{x}(n) \tag{2.32}$$

Hence, with $\beta = 1$ and $\epsilon = \frac{1}{\mu}$ we have shown that the NLMS algorithm (2.32) is equivalent to the weight update equation based on the *a posteriori error*.

2.2.c Generalised Normalised Gradient Descent (GNGD) Example for a MA(1) System

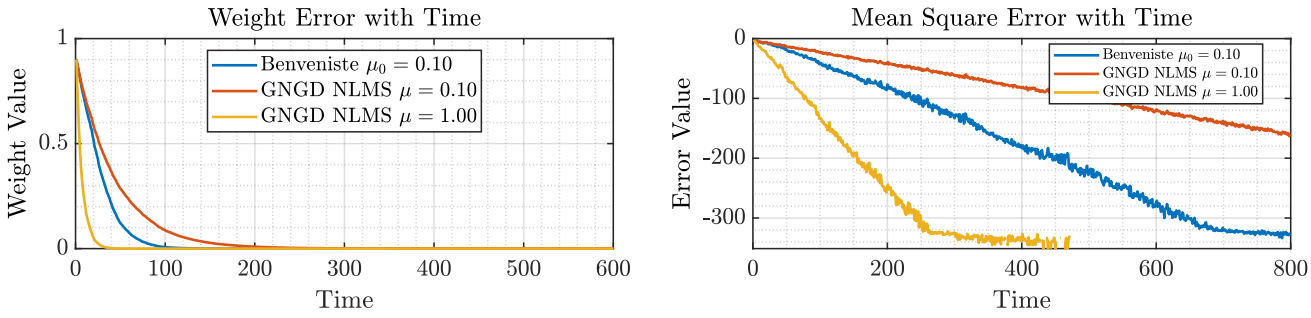


Figure 2.2.2: Weight Error and Prediction Error comparing the GNGD NLMS to the Benveniste GASS Algorithm

As we explored earlier the LMS algorithm's convergence time is closely tied to the value of μ, showing the greatest change for the $\times 10$ increase of μ. Figure 2.2.2, shows that we can surpass the rate of convergence of the Benveniste algorithm when using the NLMS algorithm. The NLMS algorithm allows us to choose large values of μ, that would traditionally cause the output of the LMS to be unstable, due to its GNGD normalisation approach.
Computationally these algorithms are driven by their outer product calculations, the Benveniste algorithm performs 1 per step, whereas the GNGD NLMS algorithm performs 3 per step making it of higher computational complexity.

2.3 Adaptive Noise Cancellation

With the: pure sine wave $x(n)$, noisy signal $s(n)$, noise term $\eta(n)$ and ALE filter output $\hat{x}(n;\Delta)$, parametrised by the delay parameter Δ. Then the Mean Squared Error (MSE) is given by:

$$\texttt{MSE} = \mathbb{E}\left[\big(s(n) - \hat{x}(n;\Delta)\big)^2\right] = \mathbb{E}\left[\big(x(n) + \eta(n) - \hat{x}(n;\Delta)\big)^2\right] \tag{2.33}$$

$$= \mathbb{E}\left[\big(\eta(n) + (x(n) - \hat{x}(n;\Delta))\big)^2\right] \tag{2.34}$$

$$= \mathbb{E}\left[\eta^2(n)\right] + \mathbb{E}\left[\big(x(n) - \hat{x}(n;\Delta)\big)^2\right] + 2\,\mathbb{E}\left[\eta(n)\big(x(n) - \hat{x}(n;\Delta)\big)\right] \tag{2.35}$$

The 1st term, noise power $\mathbb{E}[\eta^2(n)]$, is independent of Δ, the 2nd term, Mean Squared Prediction Error (MSPE) $\mathbb{E}[(x(n)-\hat{x}(n;\Delta))^2]$ is independent of noise $\eta(n)$, the 3rd term involves both the delay Δ (through $\hat{x}(n)$) the noise term and a 2 pre-factor, so is a candidate for minimisation:

$$\min_{\Delta\in\mathbb{N}} \mathbb{E}\left[\eta(n)\big(x(n) - \hat{x}(n;\Delta)\big)\right] = \min_{\Delta\in\mathbb{N}} \mathbb{E}\left[\eta(n)x(n)\right] - \min_{\Delta\in\mathbb{N}} \mathbb{E}\left[\eta(n)\hat{x}(n;\Delta)\right] \tag{2.36}$$

As $x(n)$ and $\eta(n)$ are uncorrelated, any $\mathbb{E}[\eta(n)x(n)]$ terms go to 0:

$$= \min_{\Delta\in\mathbb{N}} \mathbb{E}\left[\big(\eta(n)\big)\boldsymbol{w}^T\boldsymbol{v}(n;\Delta)\right] \tag{2.37}$$

$$= \min_{\Delta\in\mathbb{N}} \mathbb{E}\left[\big(\eta(n)\big)\sum_{i=0}^{M-1} w_i s(n-\Delta-i)\right] \tag{2.38}$$

$$= \min_{\Delta\in\mathbb{N}} \mathbb{E}\left[\big(\eta(n)\big)\sum_{i=0}^{M-1} w_i\big(x(n-\Delta-i) + \eta(n-\Delta-i)\big)\right] \tag{2.39}$$

Now using the definition of $\eta(n) = v(n) + 0.5v(n-2)$:

$$= \min_{\Delta \in \mathbb{N}} \mathbb{E}\left[\big(v(n) + 0.5v(n-2)\big) \sum_{i=0}^{M-1} w_i\big(\eta(n-\Delta-i)\big)\right] \tag{2.40}$$

$$= \min_{\Delta \in \mathbb{N}} \mathbb{E}\left[\big(v(n) + 0.5v(n-2)\big) \sum_{i=0}^{M-1} w_i\big(v(n-\Delta-i) + 0.5v(n-2-\Delta-i)\big)\right] \tag{2.41}$$

Since $v(n)$ is identically and **independently** distributed (i.i.d), *zero mean* white noise:

$$\mathbb{E}\Big[v(n)v(n-j)\Big] = 0, \quad \forall j \neq 0 \tag{2.42}$$

(2.41) is zero and minimised for $\Delta > 2$ and maximised for $\Delta = 0$, a direct consequence of (2.42). This is somewhat intuitive as the coloured noise signal $\eta(n)$ is a second order MA process. MPSE is plotted against Δ, Figure 2.3.1, verifying the improved performance for $\Delta > 2$.

2.3.a Adaptive Line Enhancer (ALE) Delay Investigation

Figure 2.3.1 shows the influence of ALE Delay on a range of model orders. We can see the minimum MPSE for Delay of 5 for Model Order 5. As we increase the delay we see a linear increase in log emphasised MPSE, letting us know that the samples that are more than 5 time points away are getting decreasingly uncorrelated to the current prediction, as expected for this second order moving average process.

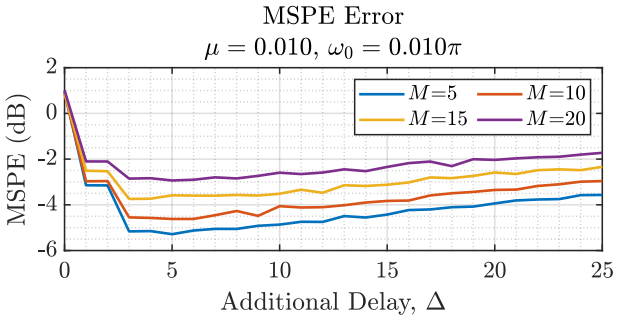


Figure 2.3.1: ALE Delay & Order Sweep

2.3.b Adaptive Line Enhancer (ALE) Filter Order Investigation

Figure 2.3.2 shows the influence of ALE Model Order for a given delay. This hyper parameter sweep is informative in understanding how many samples are needed to appropriately model the noise system, in this case we see that orders of 3 and 5 give us minimum MPSE. As we start to overfit with increasing model orders we get increasingly worse predictions.

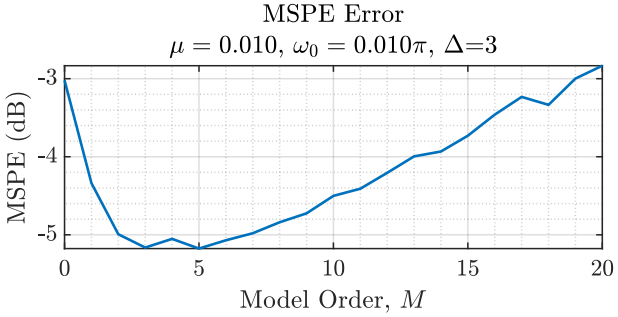


Figure 2.3.2: ALE Order Sweep

2.3.c Adaptive Noise Cancellation (ANC) Comparison

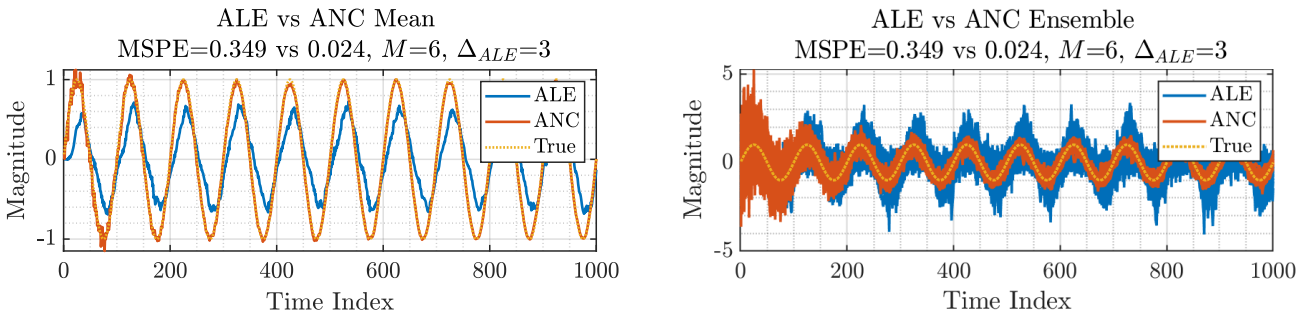


Figure 2.3.3: ALE and ANC compared, MSPE was calculated from time index 600+

To compare the ANC and ALE algorithms we fixed the Model Order to $M = 6$ and set the ALE Delay to $\Delta = 3$. This ALE Delay is one of the optimums as shown in Figure 2.3.1. What we observe in Figure 2.3.3, is the 15 times better MPSE performance of the ANC algorithm.
When looking at the ensemble of predicted traces for both algorithms we see that the ANC stabilisation time is longer than that of ALE, but the stabilised prediction is much closer to the true signal. The mean trace shows us that there is both an amplitude error and phase error for the ALE predictor, which is not seen in the ALE predictor.

2.3.d Adaptive Noise Cancellation (ANC) on real EEG Data

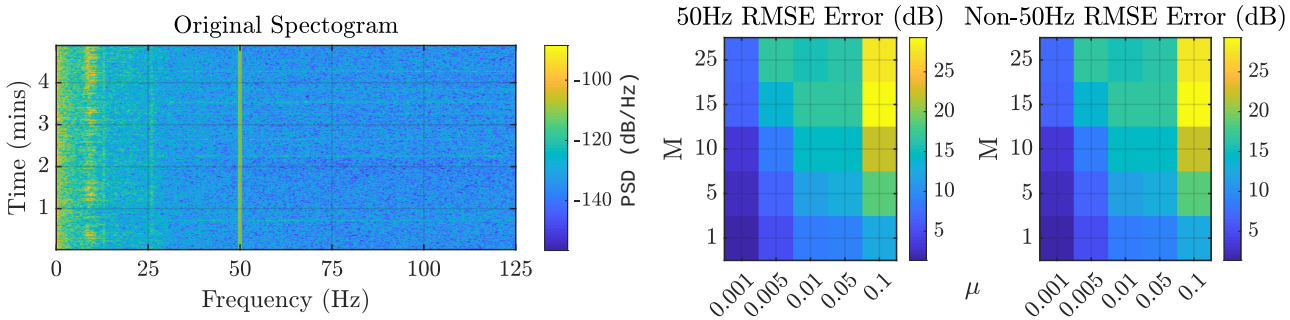


Figure 2.3.4: Original Spectrogram & investigating hyperparameters.
50Hz was defined with a ±2Hz window.

As we can see above in Figure 2.3.4, a hyper parameter heatmap highlights the RMSE tradeoffs we can expect for varying model order, M, and learning rate, μ, with this EEG dataset. Using this knowledge we selected $M = 10, \mu = 0.01$ as a satisfactory compromise.

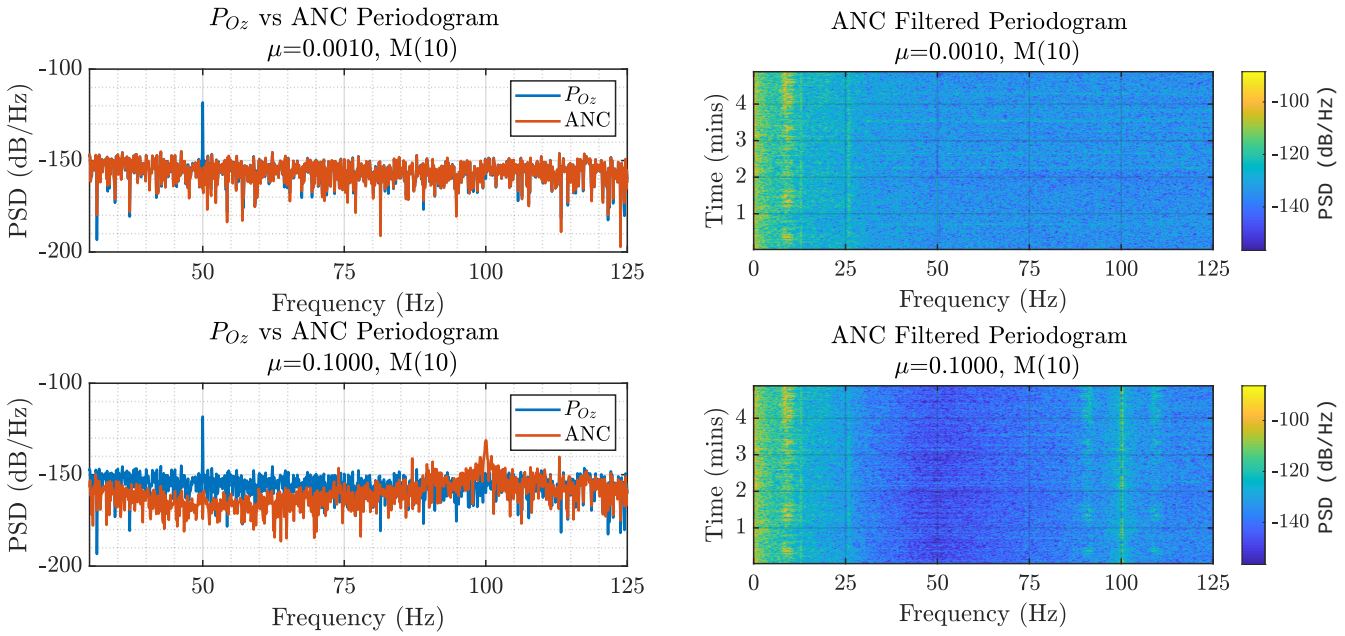


Figure 2.3.5: μ's effect on noise cancellation

In Figure 2.3.5, we can see that using the selected parameters neatly seals off the 50Hz noise. When we investigate the influence of the greater learning rate, we see great attenuation around the 50Hz band spreading by 25Hz to the surrounding frequencies. Additionally, we can see an amplification in the harmonic of the 50Hz noise, at 100Hz.

It is important to highlight the significance of the external noise fed into the ANC algorithm, testing showed that increasing the noisiness of the provided input noise signal drastically reduced ANC performance, outputs were unstable and rapidly approached infinity, becoming more prominent for larger μ. In the example in Figure 2.3.5 a 50Hz signal was generated with noise power $\sigma_\eta^2 = 0.1$. In technology where this algorithm would be useful, such as noise cancelling headphones, we would ensure that the noise external microphones pick up have a known linear relationship to what is heard inside the headphones. If this relationship is not approximately linear we can imagine that the noise cancellation effect would be less pronounced, unless other preprocessing is used before being fed into the ANC algorithm.

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

3.1.a Widely Linear Moving Average (WLMA) Process using Complex LMS (CLMS) & Augmented CLMS (ACLMS)

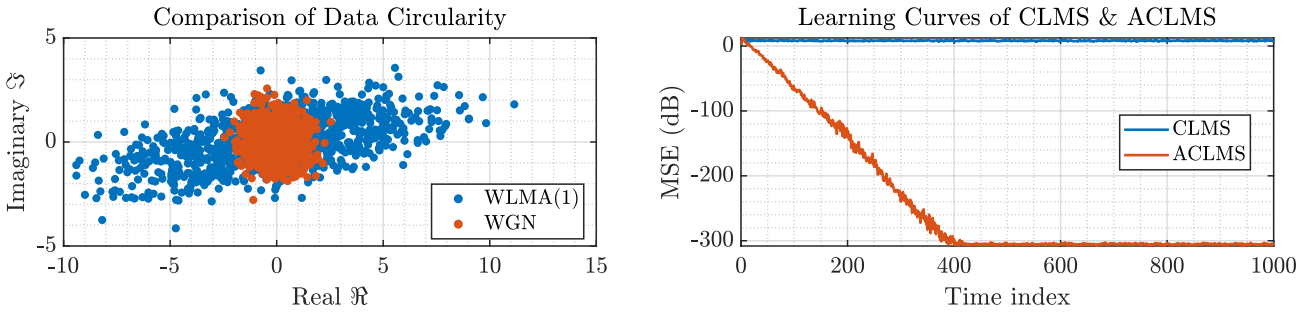


Figure 3.1.1: Data Circularity's and the suitability of CLMS & ACLMS to model a WLMA(1)

Figure 3.1.1, shows the difference in circularity of white gaussian noise and the first order widely linear moving average process. Additionally, we can see the CLMS learning curve struggle to model the non-circular WLMA(1) process, converging to an MSPE $>$ 0dB.

3.1.b CLMS and ACLMS Suitability with real (wind) data

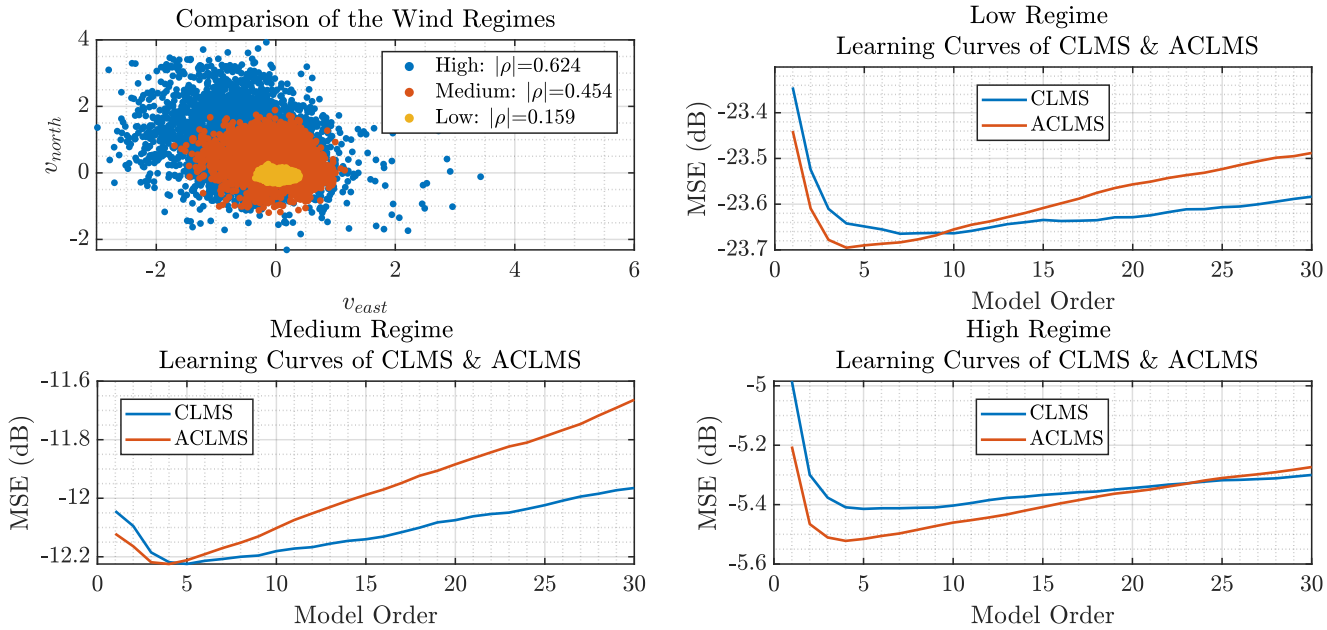


Figure 3.1.3: Wind Regime Circularity and CLMS & ACLMS estimates
Low, Medium and High Regimes used $\mu = 0.1, 0.01, 0.001$ respectively

Figure 3.1.3, shows the circularity of the various wind regimes and how the CLMS and ACLMS fare at modelling the data. This dataset highlights the rotation invariant property, where the further the regime centroid is from the origin the higher the circularity. We can observe for high circularity, low $|\rho|$, that the CLMS MSPE minimises at a larger model order compared to ACLMS. However, once circularity decreases, high $|\rho|$, the two algorithms approximately agree on model order. ACLMS tends to outperform CLMS, but we can observe for $|\rho| \approx 0.5$ with the Medium Regime the CLMS

converges to a solution that provides the same MSPE minimum. We can finally note that ACLMS consistently obtains a minimum MPSE at Model Order 4, suggesting the order of the underlying data.

3.1.c Balanced and UnBalanced Voltages - Investigating Circularity

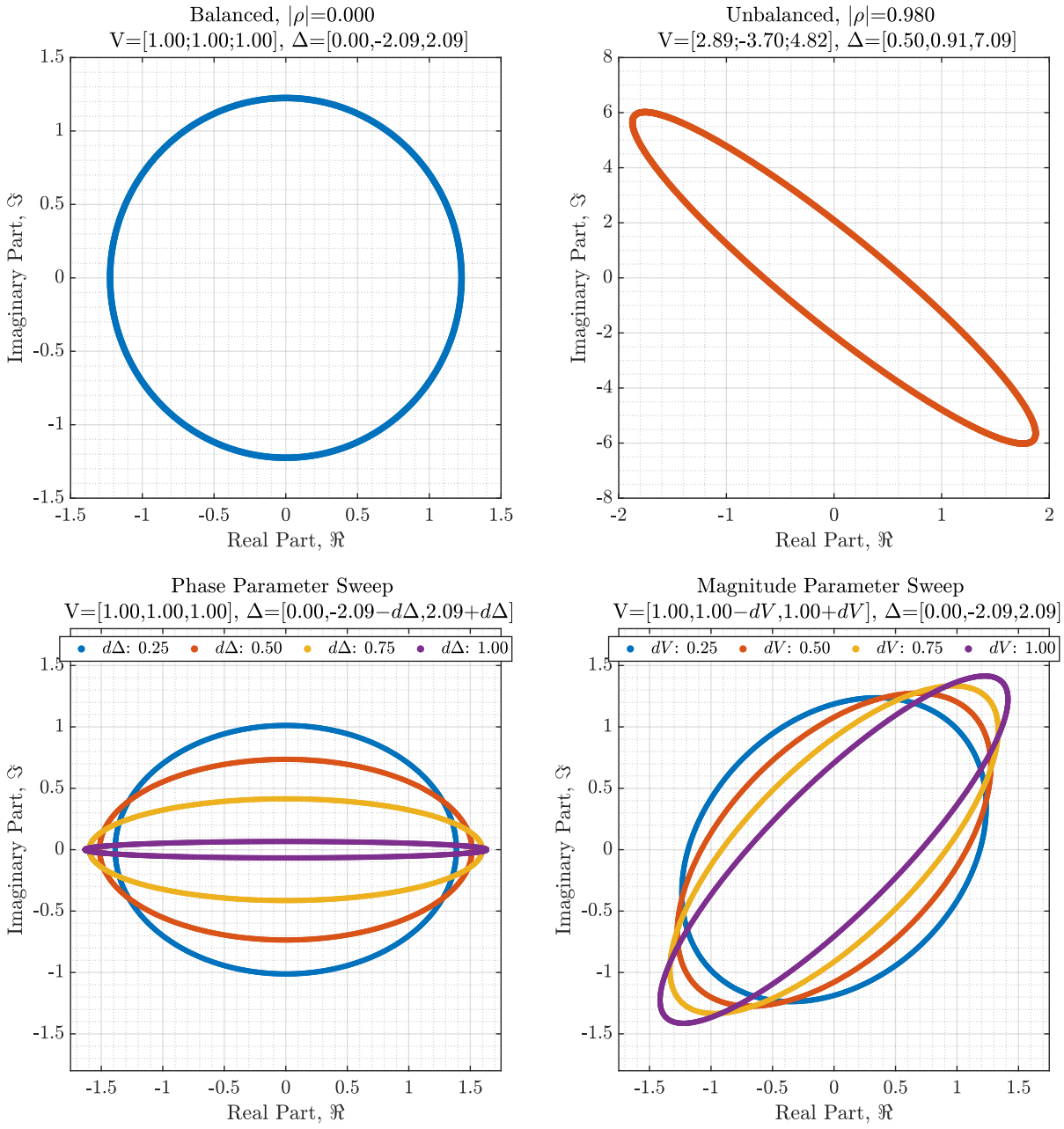


Figure 3.1.4: Voltage Data Circularity's and the influence of Phase and Magnitude Imbalance

Figure 3.1.4, demonstrates that balanced voltages are purely circular whereas unbalanced voltages are ellipses, allowing us to clearly identify if there is fault, but may be difficult to discern which bias is in play. In both cases the magnitude of the bias is proportional to the circularity, $|\rho|$.
Performing a phase sweep we observe that phase bias reduces the radius of the ellipse's minor axis and a slight change to the major axis.
In the voltage magnitude sweep we observe an increase of the ellipse's major axis radius and decrease of the minor axis radius. We can see a slight rotation about the origin as the ellipse aligns with its major axis.

3.1.d Derivation for estimating Frequency from Filter Weights

Balanced Complex Voltage Frequency

Balanced complex $\alpha-\beta$ Clarke Voltages are defined with:

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n + \phi)} \tag{3.1}$$

For time index $n+1$:

$$v(n+1) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} \tag{3.2}$$

$$= \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n + \phi)} e^{j2\pi \frac{f_o}{f_s}} \tag{3.3}$$

$$= v(n) e^{j2\pi \frac{f_o}{f_s}} \tag{3.4}$$

The Strictly Linear autoregressive model of order 1, is defined as:

$$v(n+1) = h^*(n) v(n) \tag{3.5}$$

We can express coefficient $h^*(n)$ as a function of the voltage, then take it's conjugate to give us $h(n)$:

$$h^*(n) = \frac{v(n+1)}{v(n)} = \frac{v(n) e^{j2\pi \frac{f_o}{f_s}}}{v(n)} \tag{3.6}$$

$$h^*(n) = e^{j2\pi \frac{f_o}{f_s}} \tag{3.7}$$

$$h(n) = e^{-j2\pi \frac{f_o}{f_s}} \tag{3.8}$$

$$= |h(n)| e^{j\left(arctan\left\{\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right\}\right)} \tag{3.9}$$

The general complex form, (3.9), can only be equal to the derived form, (3.8), if both their magnitudes and their angles are equal, therefore we can equate and solve for f_0:

$$2\pi \frac{f_o}{f_s} = arctan\left\{\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right\} \tag{3.10}$$

$$f_o = \frac{f_s}{2\pi} arctan\left\{\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right\} \tag{3.11}$$

Unbalanced Complex Voltage Frequency

Unbalanced complex $\alpha-\beta$ Clarke Voltages are defined with:

$$v(n) = A(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)} + B(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)} \tag{3.12}$$

The Widely Linear autoregressive model of order 1, is defined as:

$$v(n+1) = h^*(n) v(n) + g^*(n) u^*(n) \tag{3.13}$$

we substitute the Clarke Voltage Definition, (3.12):

$$v(n+1) = h^*(n)\left[A(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)} + B(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)}\right] + g^*(n)\left[A^*(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)} + B^*(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)}\right] \tag{3.14}$$

For time index $n+1$:

$$v(n+1) = A(n+1) e^{j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} + B(n+1) e^{-j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} \tag{3.15}$$

Equating (3.14) and (3.15), and by inspecting the common exponential terms:

$$A(n+1)e^{j(2\pi\frac{f_o}{f_s}(n+1)+\phi)} = \Big[h^*(n)A(n) + g^*(n)B^*(n)\Big]e^{j(2\pi\frac{f_o}{f_s}n+\phi)} \tag{3.16}$$

$$B(n+1)e^{-j(2\pi\frac{f_o}{f_s}(n+1)+\phi)} = \Big[h^*(n)B(n) + g^*(n)A^*(n)\Big]e^{-j(2\pi\frac{f_o}{f_s}n+\phi)} \tag{3.17}$$

Assuming that the amplitude change over time is negligible, $A(n+1) \approx A(n)$ and $B(n+1) \approx B(n)$, the equations simplify as follows:

$$e^{j2\pi\frac{f_o}{f_s}} = \frac{h^*(n)A(n) + g^*(n)B^*(n)}{A(n+1)} \approx h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \tag{3.18}$$

$$e^{-j2\pi\frac{f_o}{f_s}} = \frac{h^*(n)B(n) + g^*(n)A^*(n)}{B(n+1)} \approx h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \tag{3.19}$$

As (3.18) is the complex conjugate of (3.19):

$$h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} = h(n) + g(n)\frac{A(n)}{B^*(n)} \tag{3.20}$$

Multiplying with $\frac{B^*(n)}{A(n)}$ both sides:

$$\Big(h^*(n) - h(n)\Big)\frac{B^*(n)}{A(n)} + g^*(n)\left(\frac{B^*(n)}{A(n)}\right)^2 - g(n) = 0 \tag{3.21}$$

We can solve the quadratic (3.21), to find $\frac{B^*(n)}{A(n)}$:

$$\frac{B^*(n)}{A(n)} = \frac{-\Big(h^*(n) - h(n)\Big) \pm \sqrt{\Big(h^*(n) - h(n)\Big)^2 + 4g^*(n)g(n)}}{2g^*(n)} \tag{3.22}$$

$$= \frac{2j\Im\{h(n)\} \pm j\sqrt{-4\Im\{h(n)\}^2 + 4|g(n)|^2}}{2g^*(n)} \tag{3.23}$$

$$= \frac{j\Im\{h(n)\} \pm j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2}}{g^*(n)} \tag{3.24}$$

Substituting this result into (3.18):

$$e^{j2\pi\frac{f_o}{f_s}} = h^*(n) + \Im\{h(n)\}j \pm j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2} \tag{3.25}$$

$$= \Re\{h(n)\} \pm j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2} \tag{3.26}$$

$$\tag{3.27}$$

Keeping the positive solution, since $f_s \gg f_o > 0$:

$$e^{j2\pi\frac{f_o}{f_s}} = \Re\{h(n)\} + j\sqrt{\Im\{h(n)\}^2 - |g(n)|^2} \tag{3.28}$$

$$= \rho e^{j\left(\frac{\sqrt{\Im\{h(n)\}^2 - |g(n)|^2}}{\Re\{h(n)\}}\right)} \tag{3.29}$$

where $\rho > 0$.

Equating the generic, (3.9), and derived form, (3.29), then solving for f_o:

$$f_o = \frac{f_s}{2\pi} arctan\left\{\frac{\sqrt{\Im\{h(n)\}^2 - |g(n)|^2}}{\Re\{h(n)\}}\right\} \tag{3.30}$$

3.1.e Investigating the Accuracy of the Filter Estimates

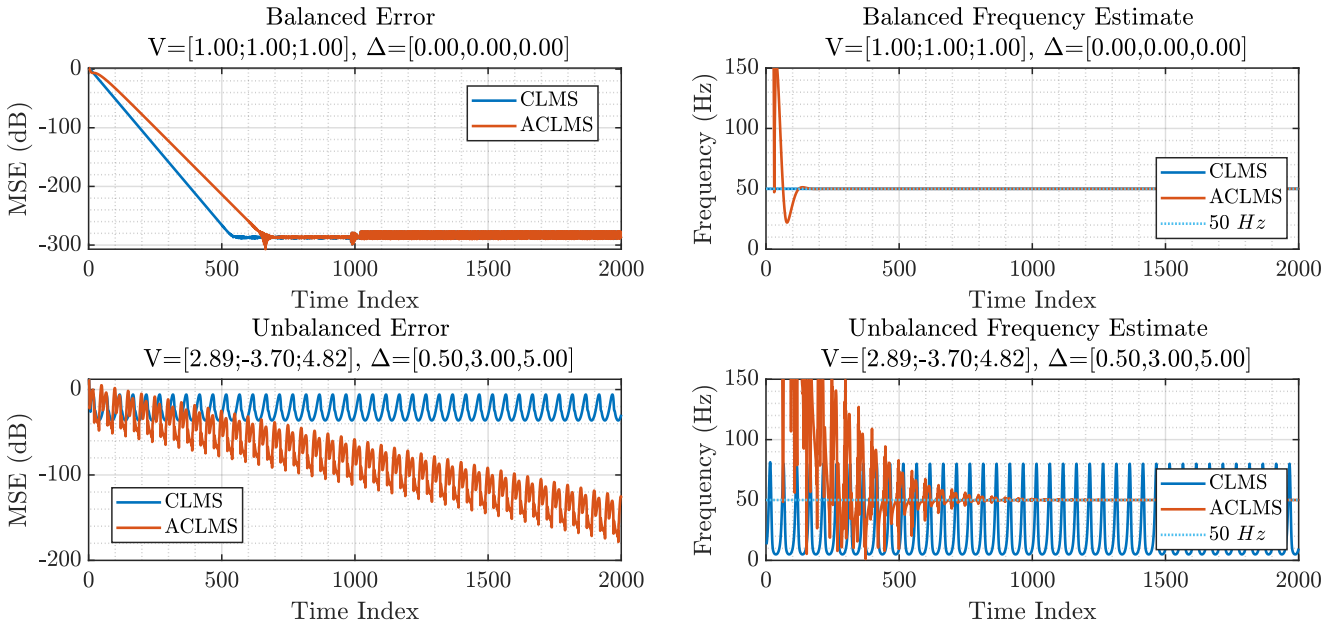


Figure 3.1.5: CLMS & ACLMS Estimates of Balanced & Unbalanced Voltage Data

In estimating a 50Hz mains noise we see that in the balanced case both algorithms converge to the correct estimate.
In the unbalanced case we see that the CLMS algorithm has troubles converging and fluctuates about the correct estimate. The ACLMS tends to converges to the correct estimate, we do note there are fluctuations in the error and in the time frame simulated and that convergence has not been reached. The discrepancy between CLMS and ACLMS in estimating the balanced and unbalanced voltages lies in the CLMS being unable to widely linear nature of the voltage and is not guaranteed to capture the complete second-order statistical relationship, as they work best with data with rotation invariant probability distributions. As was shown in Figure 3.1.4, the unbalanced voltages are no longer rotationally invariant once they form the ellipsoid (non-circular) shape.

3.2 Adaptive AR Model Based Time-Frequency Estimation

3.2.a Frequency Modulated (FM) Signal Investigated with an AR(1) Process

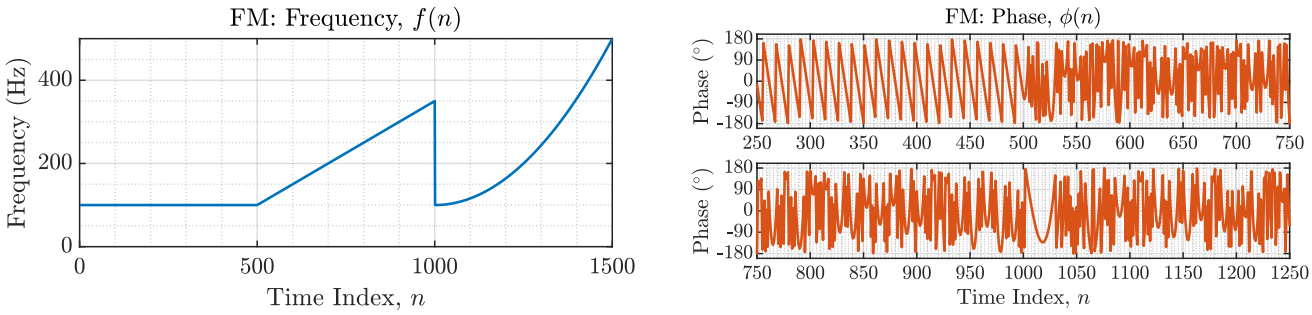


Figure 3.2.1: Input Frequency & Phase Spectrums

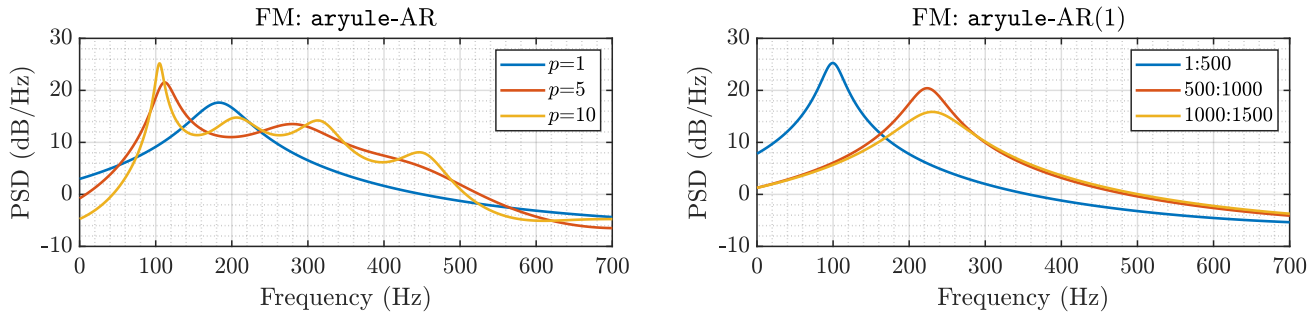


Figure 3.2.2: AR modelling of Input Data

Figure 3.2.2, highlights how AR models of varying order cannot capture the frequency variations of the signal simulated, shown in Figure 3.2.1. If we attempt to segment the signal into its piecewise parts, we see the AR(1) model only accurately identifies the 100Hz response.

3.2.b Frequency Modulated (FM) Signal Investigated with CLMS

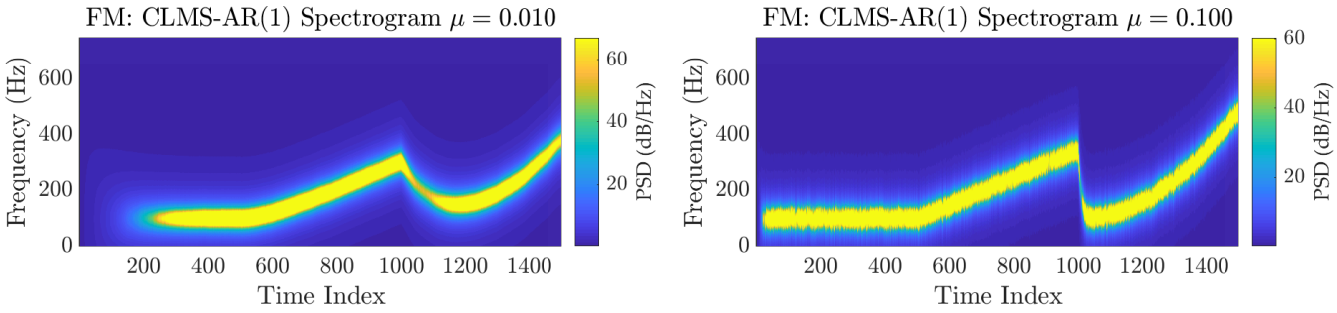


Figure 3.2.3: CLMS Spectrograms

As shown in Figure 3.2.3, we are able to better reconstruct the frequency variations using the CLMS algorithm. We must stress the importance of selecting the right μ value, we observe that we have a poor fitting curve for $\mu < 0.1$. An interesting observation is the outlier % of the total number of elements as we change μ, shown in Figure 3.2.4, we see it reaches a maximum between $\mu = 0.05$ to 0.1, corresponding to our optimum μ value range.

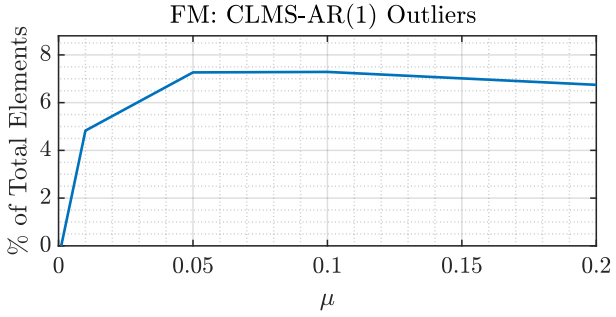


Figure 3.2.4: Outliers for Increasing μ

3.3 A Real Time Spectrum Analyser Using Least Mean Square

3.3.a The Discrete Fourier Transform (DFT) Formula and the Least Square Solution

We know that any signal, $y(n)$, can be expressed as a weighted sum of N harmonically related complex exponentials:

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k) e^{j\frac{2\pi kn}{N}} \tag{3.31}$$

We can collect all $w(k)$ weights into a vector, $\boldsymbol{w}$, and similarly all complex exponentials into a symmetric matrix, $\boldsymbol{F}$.
The predicted signal vector is thereby defined as:

$$\hat{\boldsymbol{y}} = \boldsymbol{F}\boldsymbol{w} \tag{3.32}$$

The Least Squares framework minimises the sum of square error between the true signal $y(n)$ and the estimated signal $\hat{y}(n)$:

$$\min_{\boldsymbol{w}} ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2 \tag{3.33}$$

This minimisation can be done with the gradient descent algorithm, by using the negative gradient of, (3.33).
The absolute minimum being 0, we can solve for $\boldsymbol{w}$:

$$\begin{aligned}
0 &= \nabla_{\boldsymbol{w}} \left(\min_{\boldsymbol{w}} ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2 \right) && (3.34)\\
&= \nabla_{\boldsymbol{w}} \left((\boldsymbol{y} - \boldsymbol{F}\boldsymbol{w})^H (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{w}) \right) && (3.35)\\
&= \nabla_{\boldsymbol{w}} \left(\boldsymbol{y}^H\boldsymbol{y} - \boldsymbol{w}^H\boldsymbol{F}\boldsymbol{y} - \boldsymbol{y}^H\boldsymbol{F}\boldsymbol{w} + \boldsymbol{w}^H\boldsymbol{F}^H\boldsymbol{F}\boldsymbol{w} \right) && (3.36)\\
&= -2\boldsymbol{y}^H\boldsymbol{F} + 2\boldsymbol{w}^H\boldsymbol{F}^H\boldsymbol{F} && (3.37)\\
\boldsymbol{w}^H\boldsymbol{F}^H\boldsymbol{F} &= \boldsymbol{y}^H\boldsymbol{F} && (3.38)\\
\boldsymbol{F}^H\boldsymbol{F}\boldsymbol{w} &= \boldsymbol{F}^H\boldsymbol{y} && (3.39)\\
\boldsymbol{w} &= (\boldsymbol{F}^H\boldsymbol{F})^{-1}\boldsymbol{F}^H\boldsymbol{y} && (3.40)
\end{aligned}$$

3.3.b Fourier Transform in terms of Changes of Basis & Projections

As highlighted by equation (3.40), the Fourier coefficients, i.e. $\boldsymbol{w}_{1\times N}$, are a linear combination of the (orthonormal) columns of the matrix $\boldsymbol{F}_{N\times N}$.

The DFT operation can be interpreted as a projection of the time-domain input vector, $\boldsymbol{y}$, into the frequency domain, i.e. the subspace defined by the $\boldsymbol{F}$ matrix. The columns are the basis of this new (frequency) domain, with each column representing N sinusoids, each a multiple of the frequency $\frac{f_s}{N}$, where f_s the sampling frequency.

3.3.c The DFT-CLMS vs. the AR(1) at Spectrum Analysis

Figure 3.3.1, is the spectrogram of the required DFT-CLMS. We can see a trail effect, where a frequency component is carried in time. Hence the reason this spectrogram does not match the earlier spectrogram is because the algorithm has not forgotten the previous frequency components. By implementing a leakage, forgetfulness, factor γ, we find that the CLMS can better model the underlying data.

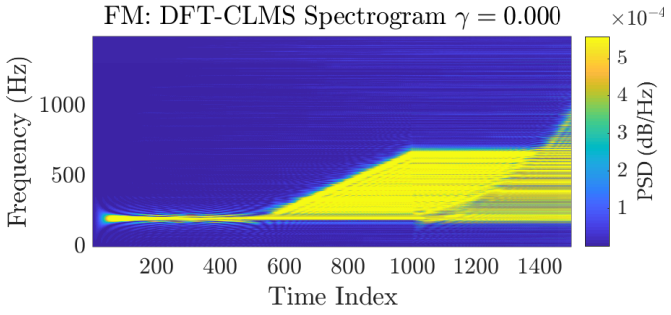


Figure 3.3.1: Effect of No γ

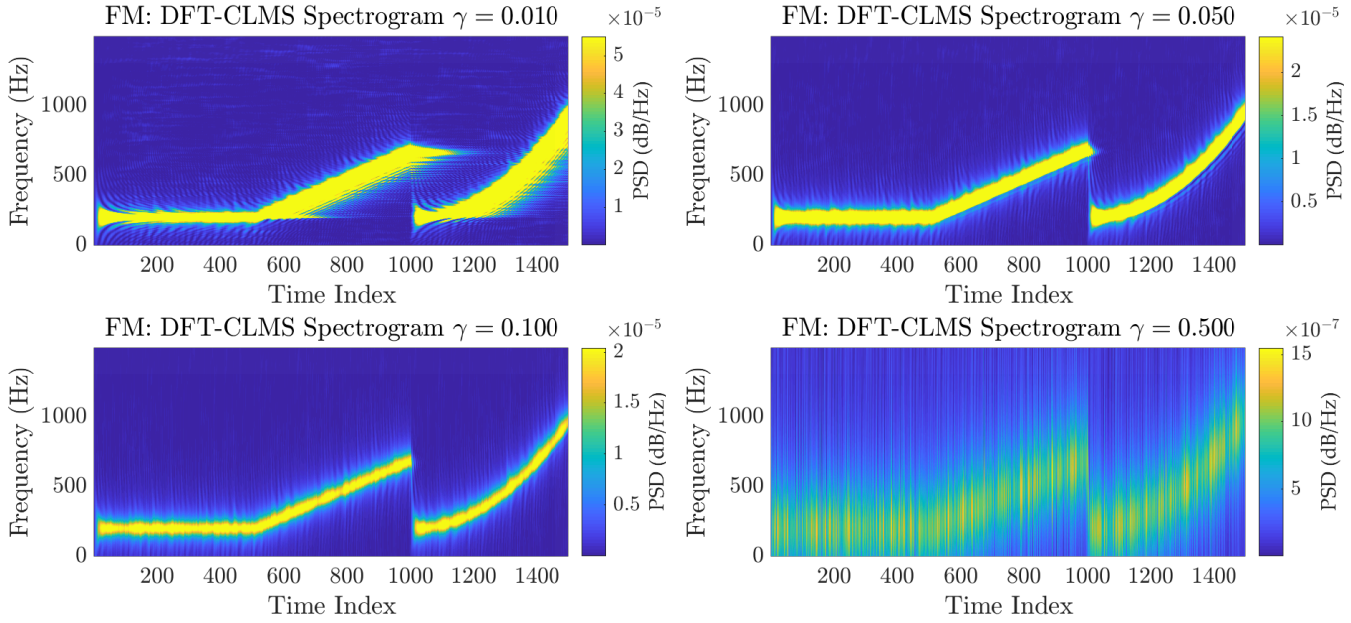


Figure 3.3.2: DFT-CLMS Spectrograms

3.3.d The DFT-CLMS on real (EEG) Data

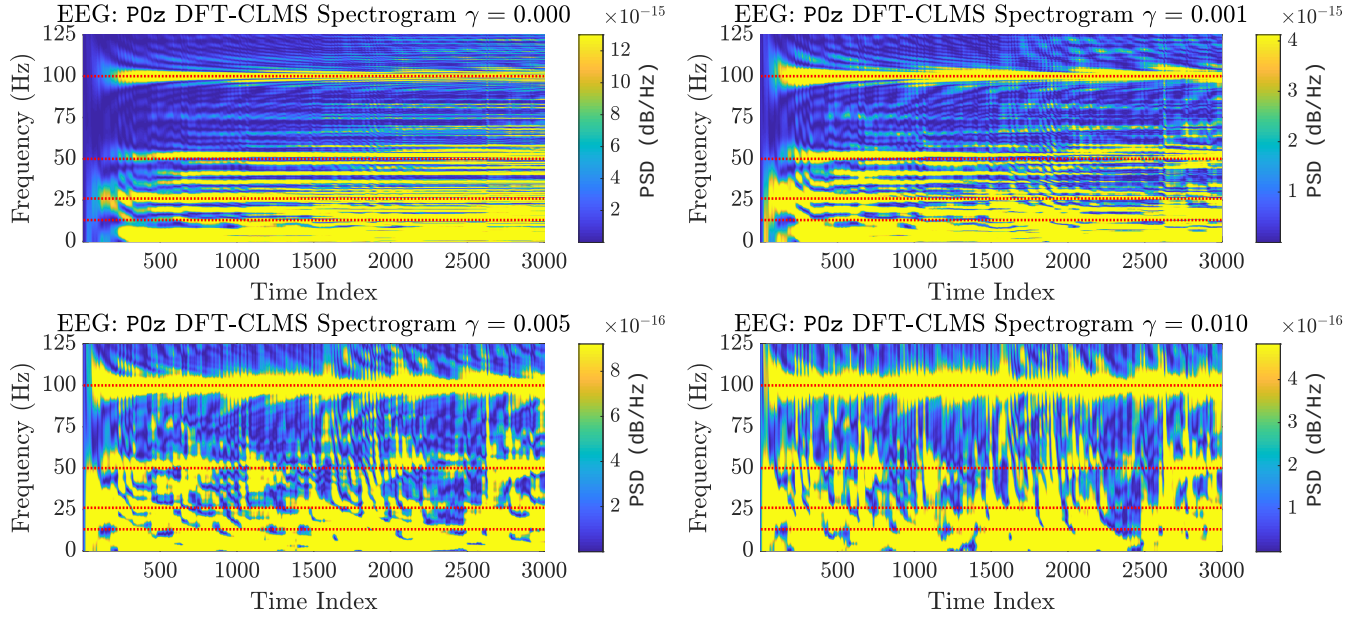


Figure 3.3.3: DFT-CLMS Spectrograms. Red lines have been plotted at: 13, 26, 50 & 100 Hz

In looking at EEG data we now find the required DFT-CLMS works better than its forgetful implementations. Frequency bands are more refined, we can better identify the SSVEP at 13 and 26Hz, the mains noise and its harmonics at 50 and and 100Hz and the alpha rhythm frequencies in the sub 10Hz range.

Why the γ factor does not improve the spectrogram accuracy can be because in the brain these frequencies are always present, hence as the algorithm starts forgetting, the error term rises and it attempts to re-learn these weights, this prevents the weights converging nicely as we see with the $\gamma = 0$ spectrogram.

4 From LMS to Deep Learning

Section	MSE	$R_p = 20\log_{10}(\frac{\hat{\sigma}_y}{\sigma_e})$
4.1	40.102	5.1957
4.2	196.74	-23.190
4.3	8.4374	13.928
4.4	Bias: 10 11.796 Bias: 1 13.600	Bias: 10 12.645 Bias: 1 12.275
4.5	10.360	14.174

Table 4.0.1: Mean Square Errors & Prediction Gains for Sections 1-5

Section	MSE	$R_p = 20\log_{10}(\frac{\hat{\sigma}_y}{\sigma_e})$
4.1	15.610	9.8044
4.2	158.53	-22.067
4.3	5.1132	15.200
4.4	Bias: 10 6.1193 Bias: 1 14.660	Bias: 10 14.582 Bias: 1 15.3295
4.5	8.7107	14.134

Table 4.0.2: Mean Square Errors & Prediction Gains for Sections 1-5. Taken from Index 500+, to remove transient effects

4.1 The LMS

In this detrended data we see the classic (linear) LMS convergence, where the amplitude starts small due to the weights initialised to zero. We consistently see the algorithm undershooting for this particular dataset, Figure 4.1.1. Ignoring the outlier from 4.2: We note that LMS has the lowest R_p value both for the entire and weight stabilised data segments, this tells us that the ratio of standard deviations is closest to 1 - showing us the variation of the predicted signal and the prediction error are most similar.

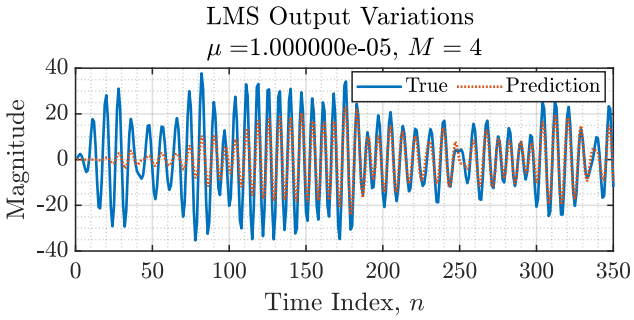


Figure 4.1.1: Linear LMS

When looking at the change from entire to weight stabilised data we can see: the LMS algorithm has the longest weight stabilisation time, ≈250 steps, reflected in it having the largest differences in MSE error and R_p.

4.2 The Dynamic Perceptron with `tanh` Activation

In this non-linear LMS convergence, the `tanh` function ensures that the output is constrained to ±1, this is clearly seen in Figure 4.2.1, with the very small amplitude. What we also observe when we look closer is the function reaches a stable prediction of sign changes of the input signal in 20 steps. The negative R_p is also a consequence of the constrained `tanh` range. As the prediction matches the fluctuation trend of the data well and the non-linearity of the activation function may help model the non-linearity of the data, we can argue that this activation is suitable for this dataset.

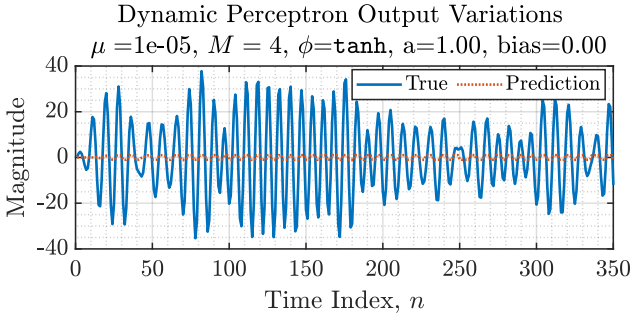


Figure 4.2.1: Dynamic Perceptron, Non-Linear LMS with `tanh` Activation

4.3 Adding Amplitude Scaling to the Dynamic Perceptron

By now scaling the magnitude of the output of the `tanh` activation we observe that the prediction now matches the true signal with much better accuracy than even that stabilised LMS. As the output range of `tanh` is ± 1, the amplitude scaling must be at least the maximum value observed in the true signal: $a \in [\mathtt{y}, \text{inf}]$ By increasing this value we would no longer need to utilise the full ± 1 range of `tanh`, this can be seen as beneficial as the closer we reach these limits the more non-linear the scaling becomes.

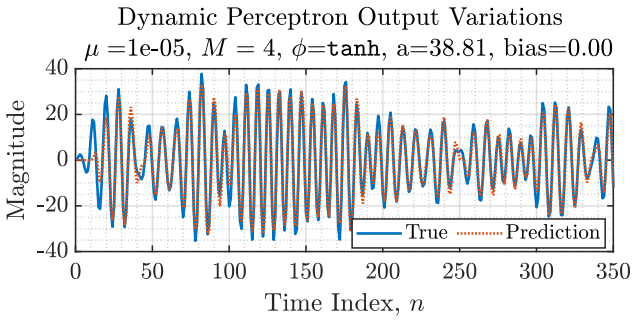


Figure 4.3.1: Dynamic Perceptron with Constant Amplitude Scaling

The R_p value is now higher than the LMS for both the entire and stabilised data ranges. This means the prediction variance is larger than our error variance, this can be interpreted as the impact of the non-linear scaling.

4.4 Biasing with the Dynamic Perceptron

Bias here can be interpreted as the sensitivity to changes in the (error) signal power, the bias value can thereby be increased to increase the sensitivity to variation in power. Using a bias of 10 instead of the proposed 1 improved MSE significantly, especially during the weight stabilised region. When looking at the convergence we see that the prediction keeps increasing in magnitude until it reaches the true signal before it begins the zero-mean prediction observed before.

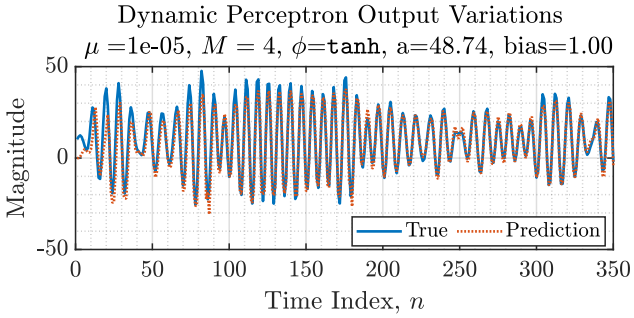


Figure 4.4.1: Dynamic Perceptron with Auto-biasing

We can see from the MSE and R_p values that the prediction is generally less accurate but has similar prediction gains. Although we should note that for a bias of 10 the prediction gain is smaller than for a bias of 1 and 0.

4.5 Pre-training the weights

Uses a single set of pre-trained weights for the entire dataset shows rapid convergence to the data mean, but we can see trade-offs by no longer using an adaptive approach, such as the increase in overshoot. This overshoot does not prevent it from outperforming the adaptive approach in MSE. Another interesting point to note is similarity of R_p to the stabilised values of a large bias - suggesting the possibility to reduce the number of epochs by using a larger bias.

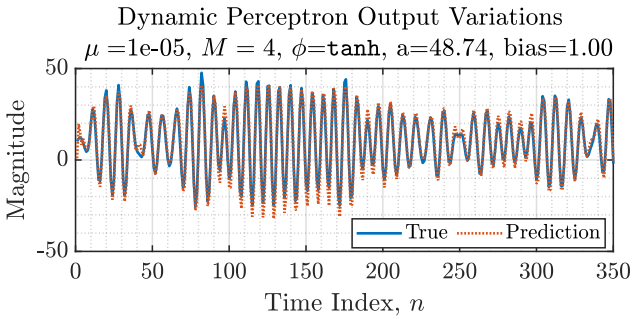


Figure 4.5.1: Dynamic Perceptron with Pre-trained Weights

4.6 What is backpropogation?

Backpropogation is an algorithm used to refine the weights used in a neural network. Its fundamental mathematic background lies within gradient descent.
The first concept we must define is a 'cost' function, $C()$. In the simplest case the cost function input is: a vector of weights (and biases) that correspond to each perceptron connection - $\boldsymbol{w}$ - and a training data set, the cost function then defines mean of the square error between the desired output and the predicted output across the training samples. This cost function has an assumed minimum at 0, which represents when both outputs are closely matched. We explicitly state the input is the set of perceptron connection weights and/or biases and the output is the mean square error for the training dataset: $C(\boldsymbol{w}) = \texttt{MSE}$
Gradient descent is the algorithm that looks to follow the negative gradient of a function, thereby finding the function **local minimum**. In this application, this means we are looking for changes to the weights in the neural network, that cause a negative change to the cost.
The backpropogation algorithm gives us the negative gradient of the cost function to follow, $-\nabla C$, which tells us how we should change the weight or bias of each connected perceptron, $\boldsymbol{w}$ to efficiently find the local minimum. Each element's magnitude represents how much the weight/bias contributes to decreasing the cost function; the sign indicates how the weight/bias should change, for example a large positive value indicates that increasing this weight/bias greatly decreases the cost function.

4.7 Deep Network Epoch Effect

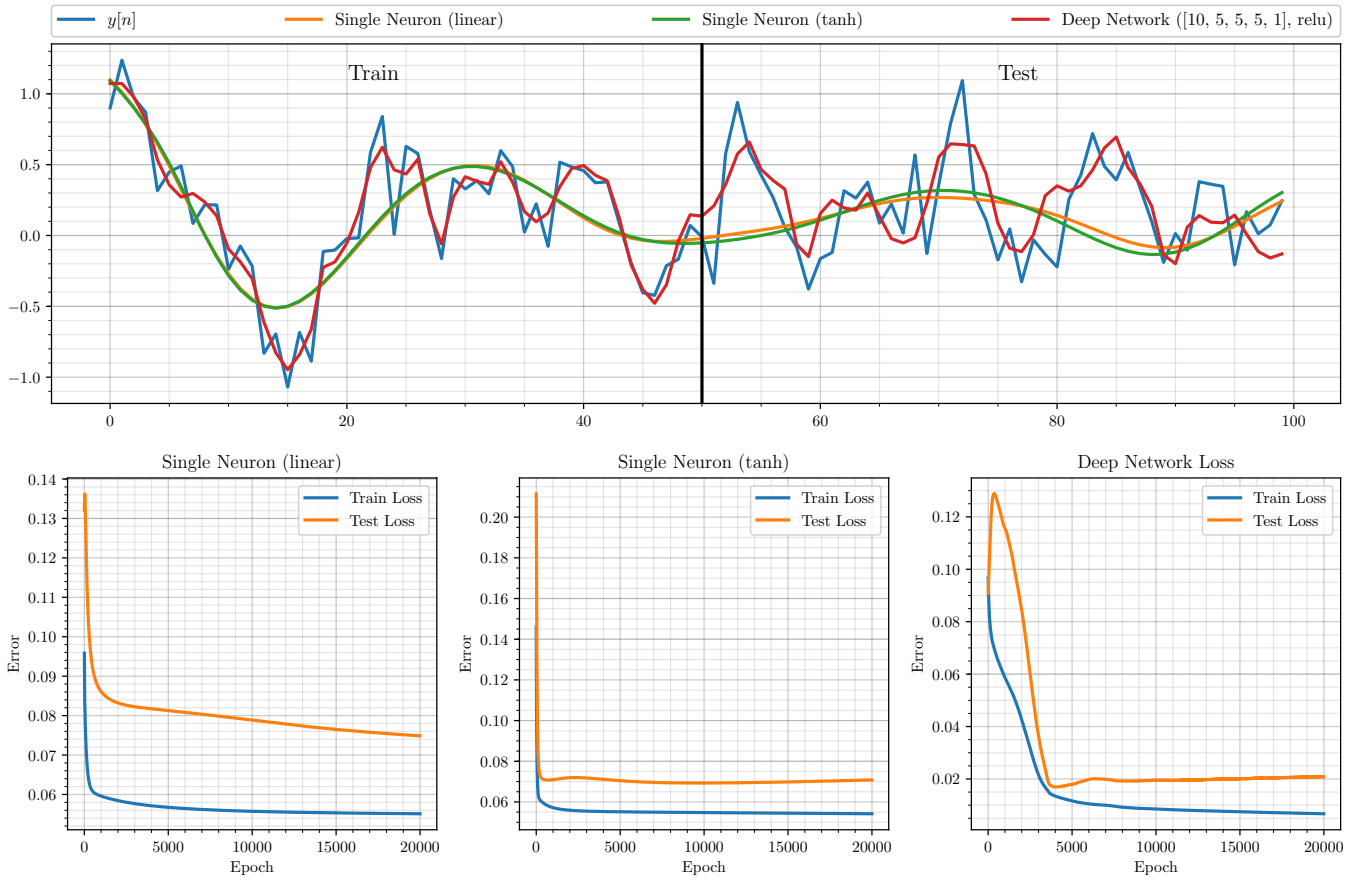


Figure 4.7.1: Learning Curves: 20,000 epochs, learning rate 0.01 and noise power 0.05

Comparing the deep network with the single dynamical perceptron, Figure 4.7.1, the deep network is better at capturing the high frequency details in the signal, whereas the dynamical perceptron has a low pass effect of the signal, seeming to capture the signal mean. This likely contributes to the lower

converged error observed in the deep network. We can additionally observe that the deep network requires a larger number of epoch steps before convergence, on both train and test data.

The number of epochs was also varied, and included in the attached Jupyter Notebook although outside the question scope, with the following observations:

For all algorithms: increasing the number of epochs tends to decrease the Test Loss. This makes sense and is equivalent to overfitting to the train data, which we see negatively impacts Test Loss for much larger epoch numbers, the algorithms most affects by this overfitting in order are: the Non-Linear LMS, Deep Network and finally the Linear LMS. When the number of epochs is too small the weights were not able to converge and provided suboptimal fitting to even the training signal magnitude. Linear LMS: Increasing the number of epochs also decreases the time to convergence until it reaches some minimum convergence time, representing the fixed learning rate. Deep network: we observe that there is a phase shift implying the algorithm is predicting the signal.

4.8 Deep Network Noise Effect

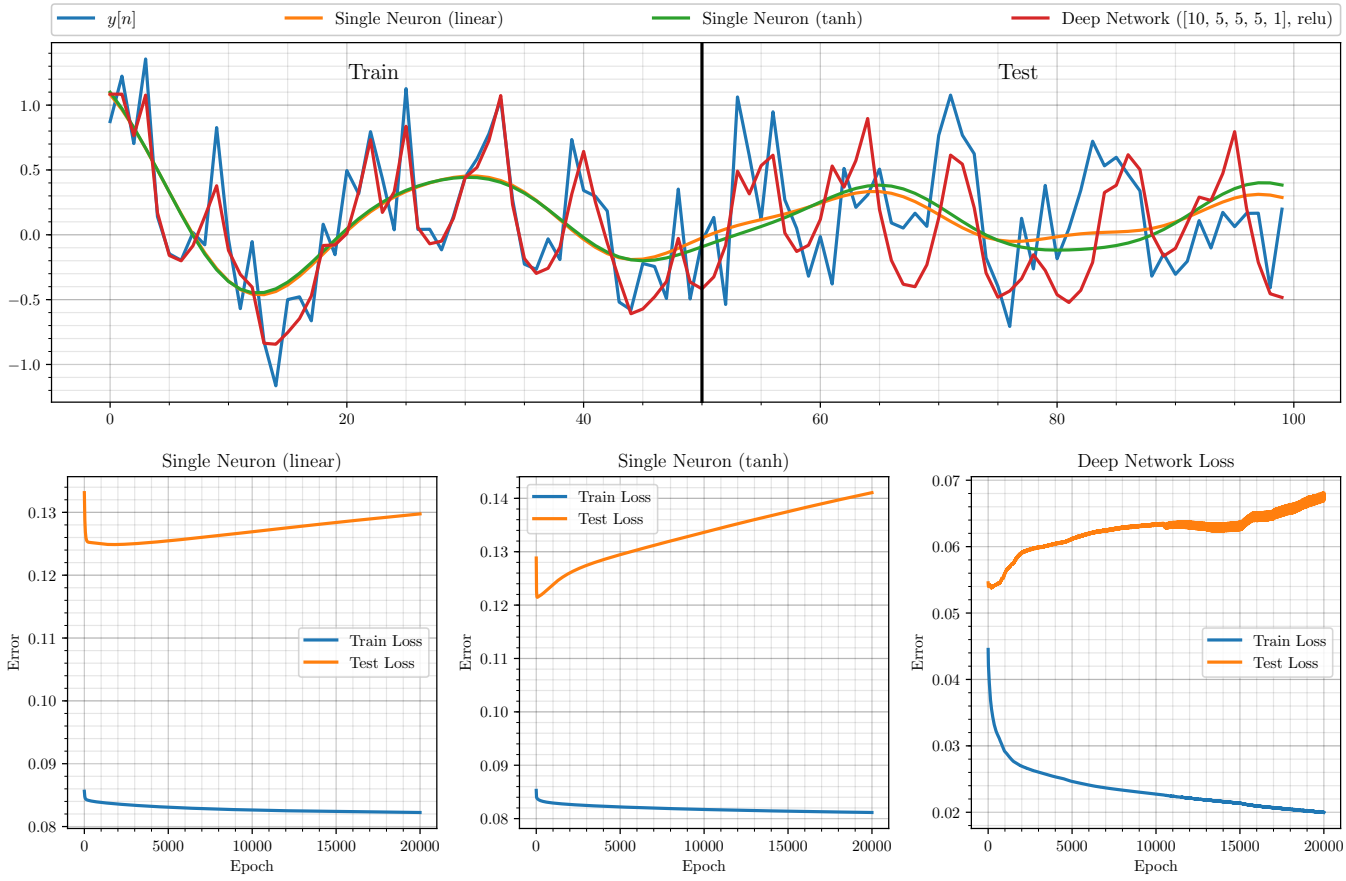


Figure 4.8.1: Learning Curves: 20,000 epochs, learning rate 0.01 and noise power 0.1

For all algorithms: increasing noise power increases Test Loss and increases the number of epochs required for convergence. Increasing the noise power also increases the likelihood that the Test Loss will not converge at all, as shown in Figure 4.8.1. This suggests that there is some characteristic of the data that is poorly modelled, reflected by how the dynamical perceptron has a sharper error gradient than the deep network. We can also observe that the deep network trace is less predictive than we saw with lower noise power in Figure 4.7.1.

Testing beyond the scope of the question, in the attached Jupyter Notebook, found that greater epochs were able to better model the test data in *low noise power regimes*, as noise power increased greater epochs simply overfitted to the test data and provided non-converging test errors.

5 Tensor Decompositions for Big Data Applications

NOTE: The interpretation of instructions has been made according to the images provided, hence a mode-1 as referred to in the images, although technically a mode-0 using Python's 0 indexing, was kept at mode-1.