Imperial College London

Imperial College London

Department of Electrical and Electronic Engineering

Adaptive Signal Processing & Machine Intelligence Coursework Report

Author: Praveen Tharmarajan

CID: 01517322

Supervisor: Prof. Danilo Mandic

April 8, 2024

Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

To begin, we were tasked with showing that the definition of PSD in Equations (1.1) and (1.2) are equivalent under the mild assumption that the covariance sequence $r(k)$ decays rapidly, i.e. Equation (1.3) is obeyed.

$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k)e^{-j\omega k} \tag{1.1}$$

$$P(\omega) = \lim_{N\to\infty} E\{\frac{1}{N}|\sum_{n=0}^{N-1} x(n)e^{-j\omega n}|^2\} \tag{1.2}$$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k||r(k)| = 0 \tag{1.3}$$

We can show this analytically by firstly expanding the definition in Equation (1.2) as follows:

$$P(\omega) = \lim_{N\to\infty} E\{\frac{1}{N}(\sum_{n=0}^{N-1} x(n)e^{-j\omega n})(\sum_{m=0}^{N-1} x(m)e^{-j\omega m})^*\} \tag{1.4}$$

$$= \lim_{N\to\infty} E\{\frac{1}{N}(\sum_{n=0}^{N-1} x(n)e^{-j\omega n})(\sum_{m=0}^{N-1} x^*(m)e^{j\omega m})\} \tag{1.5}$$

$$= \lim_{N\to\infty} E\{\frac{1}{N}(\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(n)x^*(m)e^{-j\omega(n-m)})\} \tag{1.6}$$

$$= \lim_{N\to\infty} \frac{1}{N}(\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} E\{x(n)x^*(m)\}e^{-j\omega(n-m)}) \tag{1.7}$$

$$= \lim_{N\to\infty} \frac{1}{N}(\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} r(n-m)e^{-j\omega(n-m)}) \tag{1.8}$$

The expression we obtain in Equation (1.8) is a function of $n-m$. This means that we can substitute $\alpha = n-m$ as a dummy variable, and represent the double summation as a single summation. We know that α can take values ranging from $0-(N-1) = -(N-1)$ to $(N-1)-0 = N-1$. However, since multiple combinations of n and m give rise to the same α, we must account for this when rewriting our equation. If we interpret Table 1, we can see that the number of (n,m) pairs is equal to $N-|\alpha|$.

α	No. of (n, m) pairs	(n, m) pairs
$N-1$	1	$(N-1,0)$
$N-2$	2	$(N-1,1),(N-2,0)$
$\vdots$	$\vdots$	$\vdots$
1	$N-1$	$(N-1,N-2),(N-2,N-3),\ldots$
0	N	$\vdots$
-1	$N-1$	$\vdots$
$\vdots$	$\vdots$	$\vdots$
-(N-1)	1	$(0,N-1)$

Table 1: Number of (n, m) pairs per given α

We can therefore rewrite our equation as follows:

$$= \lim_{N\to\infty} \frac{1}{N} \sum_{\alpha=-(N-1)}^{N-1} (N - |\alpha|) r(\alpha) e^{-j\omega\alpha} \tag{1.9}$$

$$= \lim_{N\to\infty} \sum_{\alpha=-(N-1)}^{N-1} r(\alpha) e^{-j\omega\alpha} - \lim_{N\to\infty} \frac{1}{N} \sum_{\alpha=-(N-1)}^{N-1} |\alpha| r(\alpha) e^{-j\omega\alpha} \tag{1.10}$$

Focusing on the second term in Equation (1.10), considering the magnitude and phase components, we can write the following inequality to define the range of magnitude:

$$0 \leq |\lim_{N\to\infty} \frac{1}{N} \sum_{\alpha=-(N-1)}^{N-1} |\alpha| r(\alpha) e^{-j\omega\alpha}| \leq \lim_{N\to\infty} \frac{1}{N} \sum_{\alpha=-(N-1)}^{N-1} |\alpha||r(\alpha)| \tag{1.11}$$

Under the assumption in Equation (1.3), the inequality is rewritten as:

$$0 \leq |\lim_{N\to\infty} \frac{1}{N} \sum_{\alpha=-(N-1)}^{N-1} |\alpha| r(\alpha) e^{-j\omega\alpha}| \leq 0 \tag{1.12}$$

Therefore, since the magnitude of the second term in Equation (1.10) is 0, the term itself is 0. Hence, we can write

$$P(\omega) = \lim_{N\to\infty} E\{\frac{1}{N} |\sum_{n=0}^{N-1} x(n) e^{-j\omega n}|^2\} = \lim_{N\to\infty} \sum_{\alpha=-(N-1)}^{N-1} r(\alpha) e^{-j\omega\alpha} = \sum_{\alpha=-\infty}^{\infty} r(\alpha) e^{-j\omega\alpha} \tag{1.13}$$

This equality can also be shown through simulations. Figure 1 shows two types of signals, accompanied by their corresponding ACFs and PSD estimates. 'Def. 1' corresponds to the PSD estimate obtained using the definition in Equation 1.1; whereas, 'Def. 2' corresponds to the PSD estimate obtained using the definition in Equation 1.2. As you can see, assuming a rapidly decaying covariance sequence, the definitions give identical results, hence equivalence holds. On the other hand, if the covariance sequence was slowly decaying, we can see that the definitions aren't equivalent as the plots aren't identical.

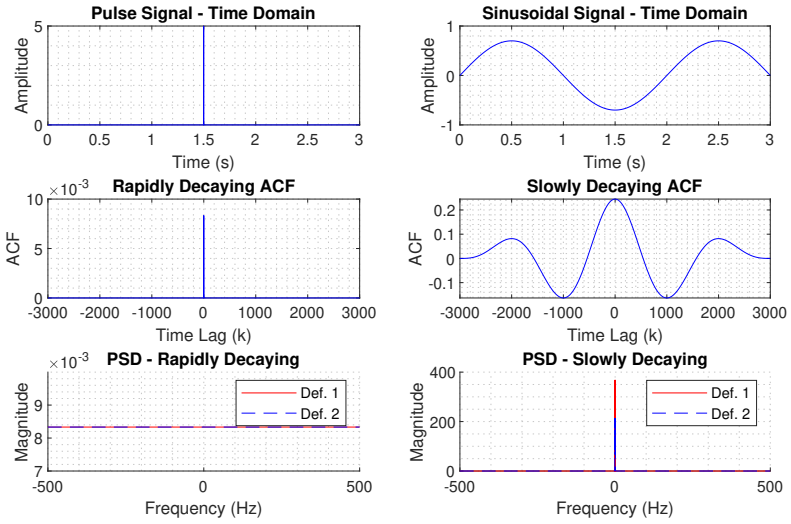


Figure 1: Two types of signals, along with their corresponding ACFs and PSD estimates showing the circumstances in which the two PSD estimates are equivalent. 'Def. 1' and 'Def. 2' corresponds to the PSD estimates obtained using Equations 1.1 and 1.2, respectively.

1.2 Periodogram-based Methods Applied to Real-World Data

(a) For this task, a periodogram-based spectral estimation technique was applied to the sunspot time-series data. The results are show in Figure 2.

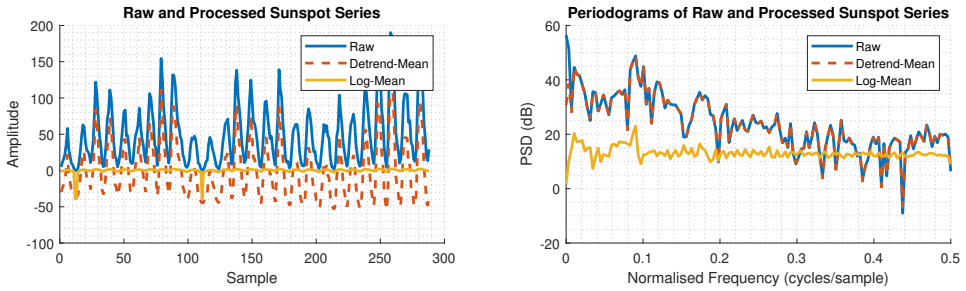


Figure 2: time-series and periodogram plots of the sunspot series

Upon removing the mean and trend components from the data, the 'Detrend-Mean' periodogram has a distinctly lower power density at $\omega = 0$, which represents the DC offset. This phenomenon is intuitive as the removal of mean centres the time-series around 0, suppressing the DC component, whilst the detrend function subtracts the best-fit line (in the least-squares sense) from the time-series. In addition, by log-transforming the data prior to preprocessing, given the monotonic feature, we observe the same periodicities at different magnitudes i.e. there are distinct peaks at the same normalised frequency as the original data. Figure 3 shows the periodogram of the log-mean sunspot series. It shows two main peaks at 0.01 cycles/samples and 0.09 cycles/sample. The peak at 0.01 cycles/sample indicates a periodicity of $\frac{1}{0.01} = 100$ samples, which is in agreement with the ACF plot, which shows a peak at a lag of 100 samples. However, the peak at 0.09 cycles/sample indicates a periodicity of $\frac{1}{0.09} = 11$ samples, which doesn't show any significant peak at a lag of 11 samples on the ACF plot.

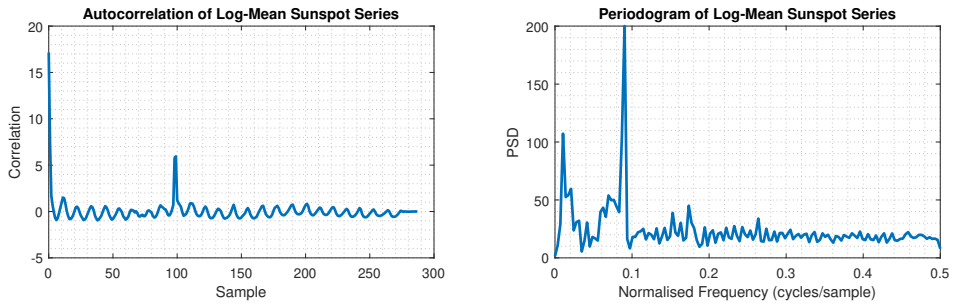


Figure 3: Autocorrelation and periodogram of log-mean sunspot series.

(b) Figure 4 draws comparison between the standard periodogram and Welch-averaged periodogram with varying window lengths. The Welch method aims to reduce noise in the estimated power spectra, in exchange for a reduction in frequency resolution. It does this by firstly splitting the potentially overlapping segments before applying a window, taking the DFT and computing the square magnitude of the result, for each segment. Each individual periodogram is then averaged to obtain a final periodogram with a reduced variance. Essentially, the Welch-averaged periodograms have a reduced variance when compared to the standard periodogram.

As the window length is changed, there is a trade-off between variance and frequency resolution i.e. a shorter window length means that there will be more segments to average, so the PSD variance will be reduced. This, however, comes at the cost of frequency resolution so it may be more difficult to differentiate narrow-band components in a signal. This can be verified by viewing Figure 4.

In all four estimates, the SSVEP response can be identified by a peak at $f_{SSVEP} = 13Hz$, with a clearly defined 2^{nd} harmonic at $f^{h2}_{SSVEP} = 26Hz$ and a visible, but not easily distinguishable 3^{rd} harmonic at $f^{h3}_{SSVEP} = 39Hz$. We can also observe the peak corresponding to the alpha-rhythm at a frequency between 8 and 10Hz, and the peak

corresponding to the power-line interference at $f_{PL} = 50Hz$.

Overall, one should note that the Welch-averaged periodograms are smoother than the standard periodogram, with the smoothness increasing as the window length is reduced. However, this smoothing effect makes peaks wider and overall less peaks can be distinguished in the case that narrow-band peaks occur. In this case, a window length of 5s gives us the best results when it comes to distinguishing $f_{SSVEP} = 13Hz$ and its harmonics from the surrounding spectra.

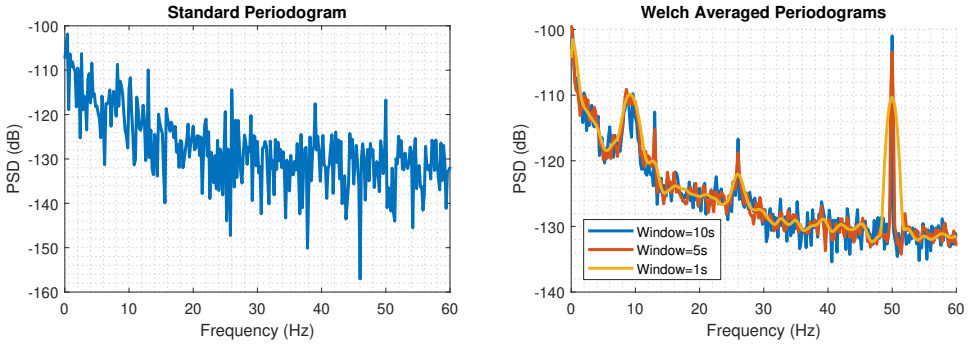


Figure 4: Standard and Welch-averaged periodograms for EEG data

1.3 Correlation Estimation

Until now, the biased sample autocorrelation has been used. The equation for the raw autocorrelation estimator, and the equations relating the biased autocorrelation estimator and unbiased autocorrelation estimator to the raw autocorrelation estimator are displayed below:

$$R_{xy}(m) = \sum_{t=k+1}^{T} (x_t - \bar{x})(x_{t-k} - \bar{x}) \tag{1.14}$$

$$R_{xy,biased}(m) = \frac{1}{T} R_{xy}(m) \tag{1.15}$$

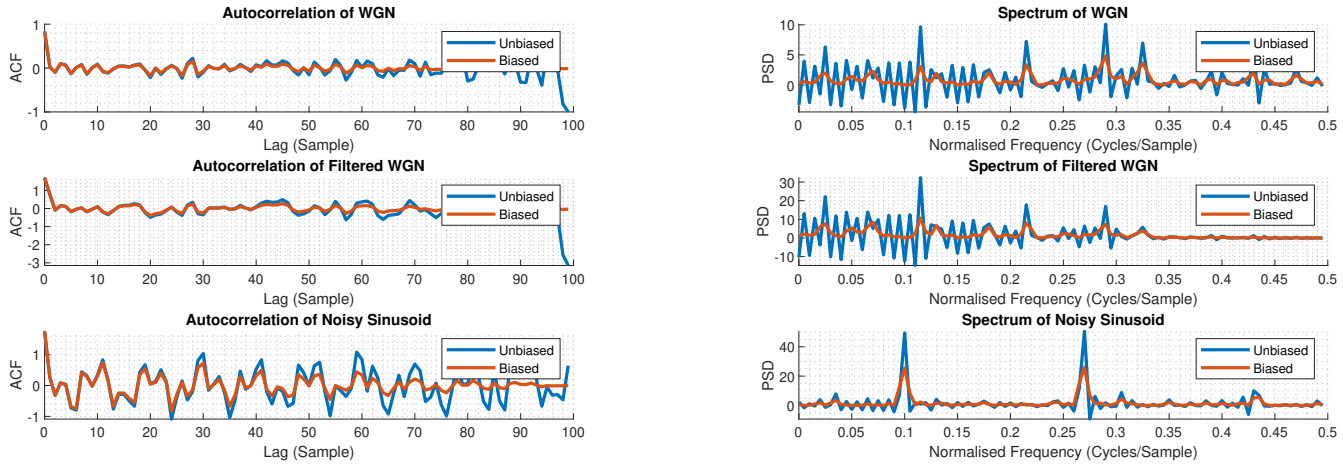
$$R_{xy,unbiased}(m) = \frac{1}{T - |k|} R_{xy}(m) \tag{1.16}$$

Essentially, the biased estimator bounds the variance by applying a Bartlett window to the unbiased autocorrelation, that is

$$R_{xy,biased}(m) = \frac{T - |k|}{T} R_{xy,unbiased}(m) \tag{1.17}$$

This has the effect of smoothing the power spectrum for statistical stability and weight suppression of the large-lag estimates. Figure 5a depicts the smoothing effect that the biased estimator has on the ACF of a signal, especially at large lags, which is a desirable characteristic. Furthermore, it can be observed that the correlations at low lags (below $k = 20$) are very similar for both the biased and unbiased estimates. However, for large lags, the unbiased and biased ACF estimates deviate from one another. Whilst the biased estimates become smoother and stable, the unbiased estimates become unstable.

Figure 5b shows the presence of some negative PSD values that were computed using the unbiased ACF estimate. It also shows a smoother curve for the biased spectra. As expected, the biased estimator doesn't suffer from negative PSD values because the Fourier transform of the Bartlett window is strictly non-negative. The combination of the smoothing effect and the non-negative PSD makes the biased estimator the better alternative of the two.



(a) Unbiased and Biased Autocorrelations

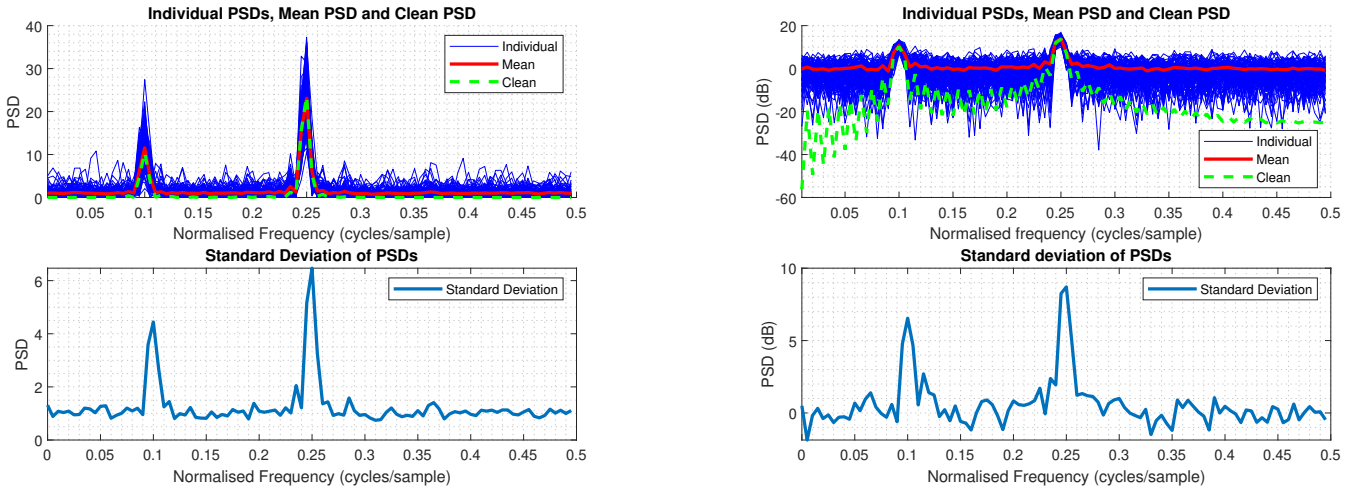
(b) Unbiased and Biased Spectra

Figure 5

Figure 6a shows the spectrum of the following signal:

$$x(n) = \sin(2\pi * 0.25n) + 0.65\sin(2\pi * 0.9n) + w(n) \tag{1.18}$$

where $w \sim \mathcal{N}(0, 1)$. We can clearly distinguish the two sine waves from the spectra by the two peaks. It is vital to note that the mean of all the corrupted signals matches the clean signal very closely. From the standard deviation graph, we can observe that the standard deviation of the corrupted signals is proportional to the PSD, meaning that most of the variance is at the frequencies of the sine waves.



(a) PSD Statistics without dB scaling

(b) PSD Statistics with dB scaling

Figure 6

Figure 6b shows how dB scaling amplifies the differences at very low PSD values whilst maintaining the general trend. Because of this scaling, the mean PSD and clean PSD have deviated from one another: the clean PSD has much more variation at frequencies that don't correspond to the sine waves. This is intuitive as theoretically, the PSD value of the clean signal should be zero at frequencies that don't correspond to the sine wave, whilst the PSD value of the corrupted signal can vary due to noise. Therefore, the dB scaling is useful when examining variations at low PSD values, typically close to the sine wave frequencies.

The periodograms in Figure 7 were computed for the following complex exponential signal:

$$z = e^{j2\pi 0.3t} + e^{j2\pi 0.32t} \tag{1.19}$$

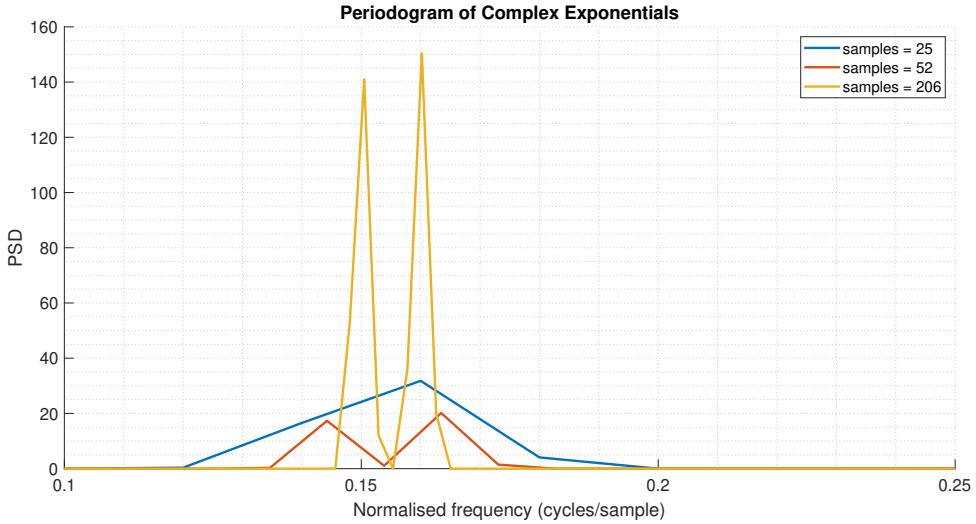


Figure 7: Periodograms of complex exponentials with differing length

The frequency components of this signal are $0.3Hz$ and $0.32Hz$. The frequency resolution of a periodogram is defined as $f_{res} = \frac{f_s}{N}$. In Figure 7, the x-axis depicts normalised frequency so the frequency resolution becomes $f_{res} = \frac{1}{N}$. Therefore, from the equation, we can deduce that at least 50 samples are required to differentiate the two frequencies present in the signal. Figure 7 also supports this claim as the periodograms of the signals with length 52 and 206 samples have clearly distinguished peaks to represent the two frequencies; whereas, the signal with a length of 25 doesn't. It's also interesting to note that the PSD falls from 25 to 52 samples and then rises from 52 to 206. Intuitively, this makes sense because, when the length is 25, a full period of the signal hasn't completed so the PSD spectrum is not representative of the whole signal. However, with 52 and 206 samples, a whole period has been complete so the periodogram can distinguish the two frequencies in the signal, as seen in Figure 8.

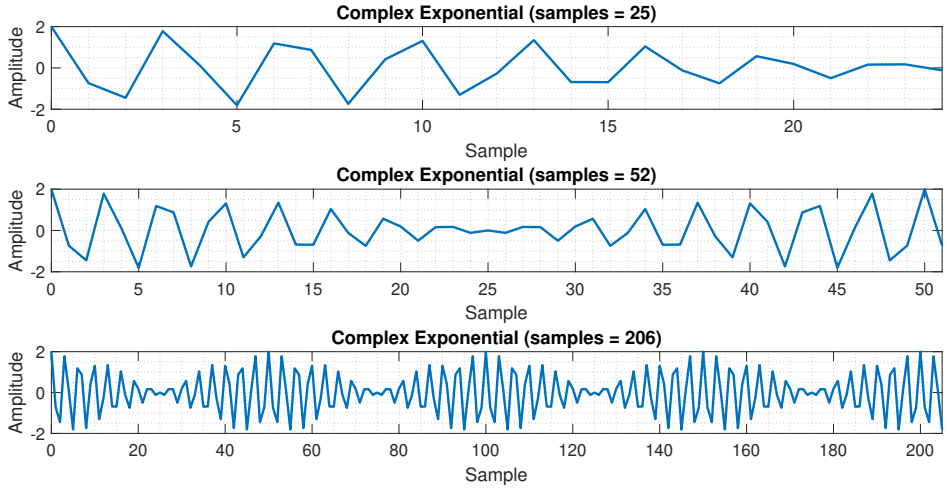


Figure 8: Real part of complex exponential signals with differing lengths

From 25 to 52 samples, the PSD values at the peaks decreases because the rise in energy from the increased length of the signal is not enough to counter the increased number of bins dividing the PSD values into smaller segments. However, when using a significantly higher number of data samples e.g. 1000 samples, the values of the PSD at the peaks are much larger because there are many more periods of the signal available, so the energy amplification at each frequency overcomes the increased number of frequency bins.

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

Figure 9

The first line of the Matlab code in Figure 9 returns the biased estimate of the autocorrelation matrix to R for an input signal x of length n. The output X is a rectangular Toeplitz matrix such that $X'X = R$. The second input to the function, m, determines the size of the biased autocorrelation matrix, which will have dimensions $(m+1)\times(m+1)$. In this case, R will be 15×15. The third input determines the method by which X is computed; 'mod' defines X as the $2(n-m)\times(m+1)$ modified rectangular Toeplitz matrix that generates an autocorrelation estimate for the input x. It is derived using forward and backward prediction error estimates, based on an m^{th}-order prediction error model.

The second line of code implements the multiple signal classification (MUSIC) algorithm, on the autocorrelation sequence provided in the matrix R, returning S, the pseudospectrum estimate of the input autocorrelation sequence, and F, the output frequencies of the spectrum. This is done by performing eigenspace analysis of the input autocorrelation matrix, R. The first input is the autocorrelation matrix. The second input specifies the signal subspace dimension. The third input specifies the normalised frequencies as a vector for which the pseudospectrum is computed at; if this has fewer than 2 elements it is interpreted as the number of NFFTs to implement. The fourth input specifies the sample rate. Finally, the last input specifies whether the first input is an autocorrelation matrix or a raw signal. In our case, it has been specified as an autocorrelation matrix. The MUSIC algorithm can be mathematically expressed as follows:

$$P_{MUSIC}(\omega) = \frac{1}{\sum_{i=p+1}^{M} |e^H v_i|^2} \tag{1.20}$$

Here, v_i represents the i^{th} eigenvector of the autocorrelation matrix, M denotes the dimension of the eigenvectors, p specifies the dimension of the signal subspace and e^H denotes a vector of complex exponentials. This algorithm demonstrates exceptional performance when processing signals that are the sum of sinusoids with Additive White Gaussian Noise (AWGN). This superiority stems from the eigenvalue decomposition of the autocorrelation matrix, where the p largest eigenvectors span the subspace of both signal and noise, while the remaining eigenvectors exclusively span the noise subspace. Consequently, the eigenvectors in the sum span the noise subspace alone. The methodology entails computing the sum of squared magnitudes of the Fast Fourier Transform (FFT) for each noise-spanning eigenvector. Despite offering enhanced resolution and resilience in noisy conditions compared to the FFT, this approach incurs substantially higher computational overheads, characterised by a time complexity of $\mathrm{O}(N^3)$ as opposed to $\mathrm{O}(N\log_2 N)$. Furthermore, successful application necessitates a prior understanding of the number of signal components.

The third line of MATLAB code plots the pseudospectrum and restricts the frequency representation to the range between 0.25Hz and 0.4Hz.

Figure 10 depicts the mean and standard deviation of the MUSIC spectrum for the complex signal (1.19) with AWGN added. Across all sample cases, there exists a strong correlation between the standard deviation and the mean. Specifically, with 17 samples, the MUSIC algorithm fails to differentiate between the two signal frequencies; however, with 26 and 98 samples, a clear distinction emerges. This is an improvement over the standard FFT method, which necessitated a signal of at least 50 samples to achieve satisfactory frequency resolution.

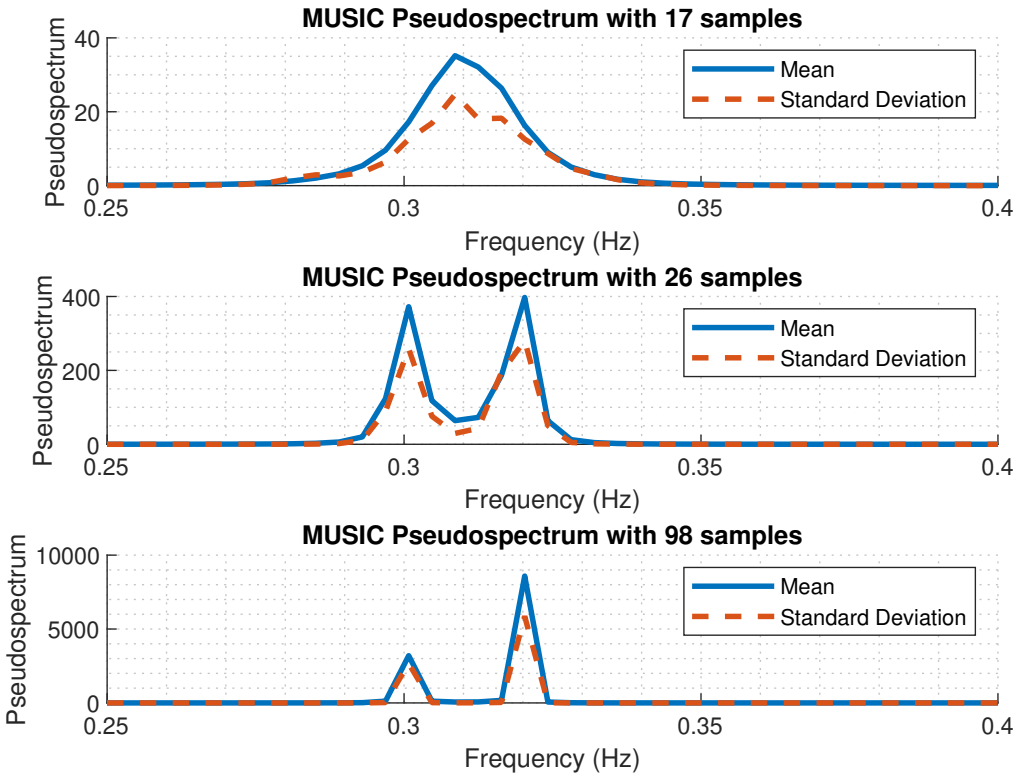


Figure 10: Mean and standard deviation of the MUSIC spectrum generated by noisy complex exponentials

1.4 Spectrum of Autoregressive Processes

For the Yule-Walker equation to hold when finding the AR parameters, it is imperative that the autocorrelation matrix be invertible. The desired relationship is shown below:

$$a = R^{-1}r \tag{1.21}$$

Here, a denotes the AR coefficients, R denotes the autocorrelation function and r corresponds to the product of the Kronecker delta function and the noise variance. The autocorrelation matrix is inherently symmetric by definition. Positive definiteness serves as a sufficient condition for a symmetric matrix to be invertible. In the case where the autocorrelation matrix is defined as biased, it meets this condition. However, in the unbiased scenario, this condition may no longer hold true. Consequently, the unbiased estimate for the ACF proves unsuitable when seeking the AR coefficients.

1000 samples and 10000 samples were generated from the AR(4) process defined by Equation (1.22).

$$x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n) \tag{1.22}$$

where $w \sim \mathcal{N}(0, 1)$.

PSD estimates of various orders of AR estimates were computed for both 1000 and 10000 samples, and the results are depicted in Figure 11.

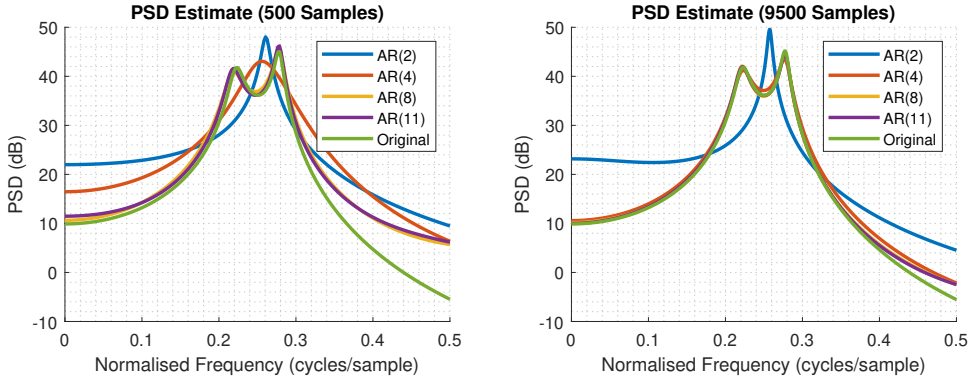


Figure 11: AR process estimation using 500 samples (left) and 9500 samples (right).

From the left-hand graph in Figure 11, it is evident that with 500 samples utilised to estimate the AR spectral estimate, when under-modelling (refer to AR(2)), it was impossible to distinguish between the two peaks. However, more complex estimates at higher model orders (refer to AR(8) and AR(11)) were able to discern between the two peaks, albeit at the expense of higher computational complexity and overfitting. There are two interesting points to note from this graph:

1. Using 500 samples, AR(4) (the true model order) was unable to discern between the two peaks accurately.
2. None of the model orders accurately estimated the PSD at higher frequencies using 500 samples.

Conversely, when employing 9500 samples, the 4^{th} order estimate exhibited improved performance in discerning the two peaks, nearly matching the performance of the higher model orders. Additionally, all model orders provided better estimates for the higher frequencies of the PSD.

1.5 Robust Regression

If we examine the left subplot of Figure 12, we can observe 3 non-zero Singular Values (SVs) for the clean signal and 10 non-zero SVs for the noisy signal, implying that the rank of $\mathbf{X}$ is 3, whilst the rank of $\mathbf{X_{noise}}$ is 10. As seen in the right subplot of Figure 12, the squared error between the singular values becomes notably larger for components in the noise subspace. This observation highlights how noise affects the singular values, making them less distinguishable and potentially misleading in identifying the true rank of $\mathbf{X_{noise}}$. Particularly, as the noise magnitude increases, the singular values of the noise subspace approach the magnitude of those in the signal subspace, further complicating the task of rank identification.

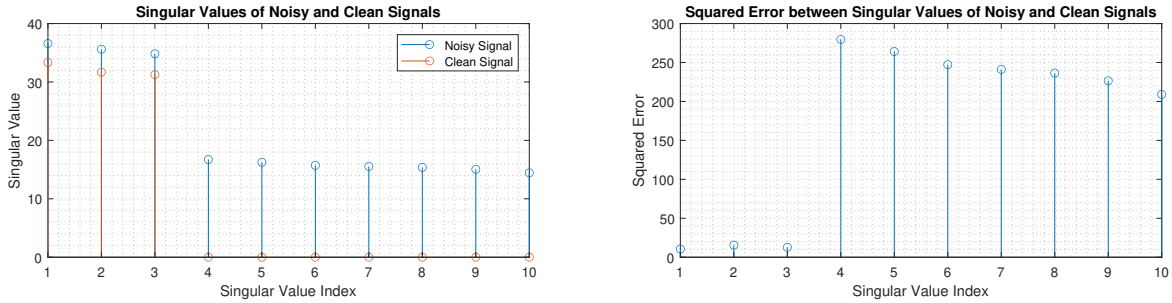


Figure 12: SVs of $\mathbf{X}$ and $\mathbf{X_{noise}}$ (left), and corresponding squared error between their SVs (right)

Using only the 3 most significant principal components, as determined by the rank of $\mathbf{X}$, a low-rank approximation of $\mathbf{X_{noise}}$, denoted by $\tilde{\mathbf{X}}_{\mathbf{noise}}$ was generated. The MSE was then computed between each column of $\mathbf{X}$ and both $\mathbf{X_{noise}}$ and $\tilde{\mathbf{X}}_{\mathbf{noise}}$.

As depicted in Figure 13, the MSE values for ($\mathbf{X}$,$\mathbf{X_{noise}}$) are considerably higher across all columns compared to those for ($\mathbf{X}$,$\tilde{\mathbf{X}}_{\mathbf{noise}}$). This notable difference in MSE is further illustrated by the total MSE values, with MSE($\mathbf{X}$,$\mathbf{X_{noise}}$)=0.2435 and MSE($\mathbf{X}$,$\tilde{\mathbf{X}}_{\mathbf{noise}}$)=0.0733.

This approach allows us to observe how the low-rank approximation improves the fidelity of the noisy signal $\mathbf{X_{noise}}$ compared to the original signal $\mathbf{X}$, reducing the overall error. However, it's essential to note that while $\tilde{\mathbf{X}}_{\mathbf{noise}}$ provides a better approximation to $\mathbf{X}$ than $\mathbf{X_{noise}}$, there's still some residual error, indicating the presence of noise components in the denoised signal. Nonetheless, the technique effectively demonstrates the utility of low-rank approximation in noise reduction.

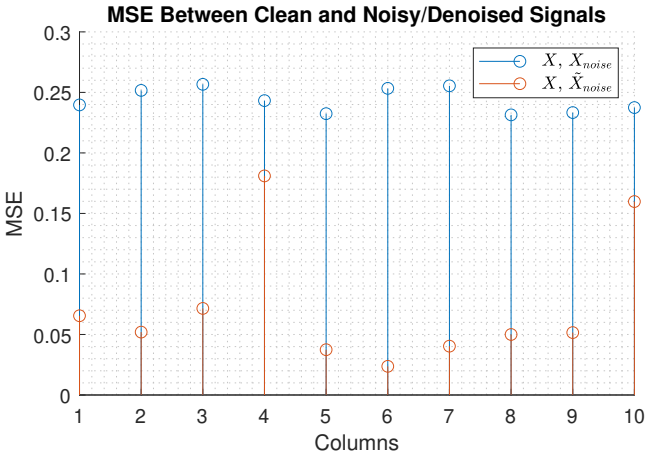


Figure 13: MSE comparison between clean signal and noisy signal before and after low-rank approximation

The Ordinary Least Squares (OLS) and Principal Component Regression (PCR) are two different regression methods. The OLS solution for the parameter matrix $\mathbf{B}$ is given by:

$$\mathbf{B}_{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

The PCR solution, considering the $r = 3$ principal components identified in the signal space, is given by:

$$\mathbf{B}_{\text{PCR}} = \mathbf{V}_{1:r}{\boldsymbol{\Sigma}_{\mathbf{1:r}}}^{-1}{\mathbf{U}_{\mathbf{1:r}}}^T\mathbf{Y}$$

After obtaining these solutions, their performances were compared using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) for both estimation and test errors. Here are the results:

- OLS:
  - RMSE (estimation) = 0.847
  - MAE (estimation) = 0.658
  - RMSE (test) = 0.689
  - MAE (test) = 0.528
- PCR:
  - RMSE (estimation) = 0.846
  - MAE (estimation) = 0.658
  - RMSE (test) = 0.686
  - MAE (test) = 0.525

As observed, both OLS and PCR methods exhibit similar performances for the given data. Despite minor differences in RMSE and MAE values, both methods provide effective solutions for estimating the regression coefficients from the noisy data $\mathbf{X_{noise}}$. The PCR method, however, offers a slightly lower total error of 2353.9 on the test set in comparison to the OLS method's total error of 2377.4. Thus, the PCR method performs slightly better than the OLS method in this scenario.

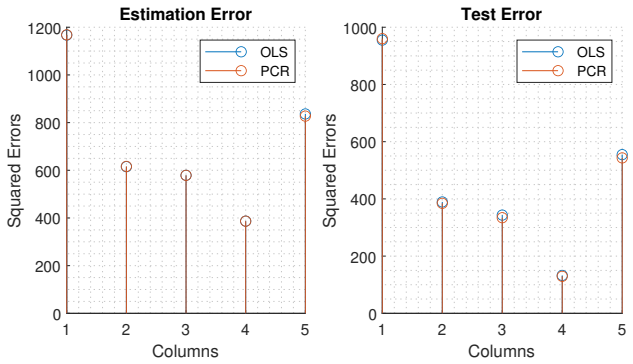


Figure 14: Comparison of OLS and PCR methods

Using the `regval` script that was provided, 1000 new realisations of the test data $\mathbf{Y}$, and its estimate $\tilde{\mathbf{Y}}$ were generated. The MSE estimates for the PCR and OLS schemes were computed and are illustrated in Figure 15. Upon analysing numerous experimental trials, it is evident that the PCR method exhibits approximately a 1% improvement over the OLS method.

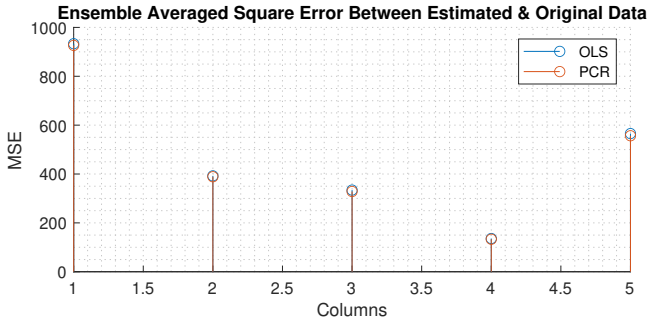


Figure 15: Comparison of OLS and PCR methods (1000 realisations)

2 Adaptive Signal Processing

2.1 The Least Mean Square (LMS) Algorithm

In the context of adaptive filters, the Least Mean Squares (LMS) algorithm serves as a dynamic system, wherein inputs $x(n-1)$ and $x(n-2)$ yield outputs representing estimates $\hat{a}_1$, $\hat{a}_2$, and $\hat{x}(n)$. This framework enables adaptation to non-stationary signals, crucial for real-world applications.

Given an Autoregressive of order 2 (AR(2)) process:

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

where η represents zero-mean Gaussian noise with variance σ_η^2, the Autocorrelation Function (ACF), $r_{xx}(k)$, can be derived as follows:

$$r_{xx}(k) = \begin{cases} \sigma_x^2 & \text{for } k = 0 \\ \text{cov}(x(n), x(n-k)) & \text{for } k > 0 \end{cases} \tag{2.2}$$

Expanding the expression yields

$$r_{xx}(k) = a_1 r_{xx}(k-1) + a_2 r_{xx}(k-2) + \delta(k)\sigma_\eta^2 \tag{2.3}$$

where $\delta(k)$ is the Kronecker delta function.

Furthermore, the cross-correlation term $E[\eta(n)x(n-k)]$ is given by

$$E[\eta(n)x(n-k)] = \begin{cases} \sigma_\eta^2 & \text{for } k = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Utilising these expressions, we derive the correlation matrix, R, of the input vector $x(n) = [x(n-1), x(n-2)]^T$ as

$$R = \begin{pmatrix} \sigma_x^2 & r_{xx}(1) \\ r_{xx}(1) & \sigma_x^2 \end{pmatrix} \tag{2.5}$$

where σ_x^2 and $r_{xx}(1)$ are computed as

$$\sigma_x^2 = \frac{\sigma_\eta^2(1-a_2)}{(1+a_2)((1-a_2)^2 - a_1^2)} \tag{2.6}$$

$$r_{xx}(1) = \frac{a_1 \sigma_x^2}{1-a_2} \tag{2.7}$$

Given specific parameter values $a_1 = 0.1$, $a_2 = 0.8$, and $\sigma_h^2 = 0.25$, we get

$$\sigma_x^2 = 0.926 \tag{2.8}$$

$$r_{xx}(1) = 0.463 \tag{2.9}$$

The resulting correlation matrix is

$$R = \begin{pmatrix} 0.926 & 0.463 \\ 0.463 & 0.926 \end{pmatrix} \tag{2.10}$$

For convergence of the LMS algorithm in the mean, the step size μ must satisfy

$$0 < \mu < \frac{2}{\lambda_{\max}} \tag{2.11}$$

where $\lambda_{\max}$ denotes the largest eigenvalue of the correlation matrix R. Given eigenvalues $\lambda_1 = 0.463$ and $\lambda_2 = 1.389$, the permissible step size range is

$$0 < \mu < 1.44 \tag{2.12}$$

Overall, the LMS algorithm, when applied to an AR(2) process, demonstrates adaptability to varying signals, with convergence ensured within the specified step size range.

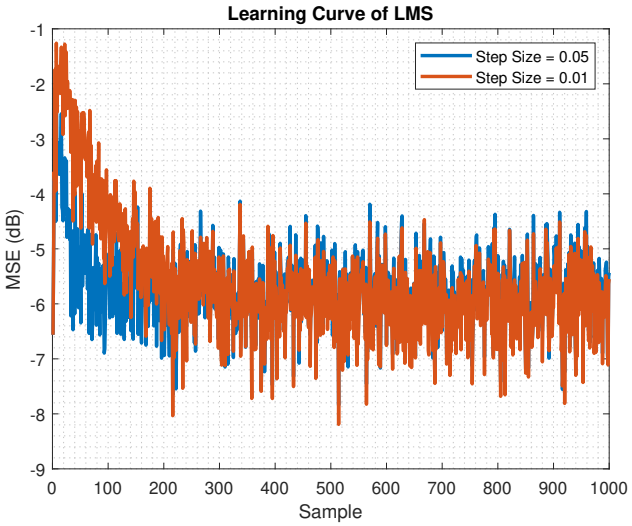


Figure 16: LMS learning curve of AR(2) process for step sizes $\mu = 0.05$ and $\mu = 0.01$

.

Figure 16 presents learning curves depicting the performance of an LMS predictor for a given model, represented by Equation (2.1), under different step sizes, specifically, $\mu = 0.01$ and $\mu = 0.05$. The larger step size, $\mu = 0.05$, clearly leads to faster convergence of the error, albeit at the cost of a larger steady-state error. These observations align with the fundamental characteristics of the LMS algorithm as a gradient descent approach.

The accelerated convergence observed with larger step sizes can be attributed to the algorithm taking larger steps down the error curve, thus approaching the minima more rapidly. However, at steady-state, the larger step sizes tend to induce oscillations around the minimum point, with the magnitude of deviation being proportional to the step size. Moreover, excessively large step sizes can lead to overshooting, potentially causing divergence from the local minima.

The misadjustment in LMS adaptive filters is quantified as the ratio of the Excess Mean Square Error (EMSE) to the minimum achievable mean square error. Specifically, $M = \frac{EMSE}{\sigma_\eta^2}$. Additionally, an approximation for the misadjustment in LMS filters is given by $M_{LMS} \approx \frac{\mu}{2}\mathrm{Tr}(R)$, where μ represents the step size and R is the autocorrelation matrix of the input signal.

The LMS algorithm was applied to the AR(2) process defined in (2.1) across 100 independent trials. By time-averaging over the steady-state of the ensemble-averaged learning curves, I was able to gain key insights into performance metrics, including EMSE and misadjustment, across varying step sizes. Table 2 presents some key numbers for step sizes $\mu = 0.01$ and $\mu = 0.05$.

μ	EMSE	M	M_{LMS}	Misadjustment Error
0.01	0.0026	0.0103	0.0093	9.71%
0.05	0.0130	0.0521	0.0463	11.1%

Table 2: Key numbers for step sizes $\mu = 0.01$ and $\mu = 0.05$.

The experimental results align with theoretical expectations, demonstrating a decreasing trend in misadjustment with smaller step sizes.

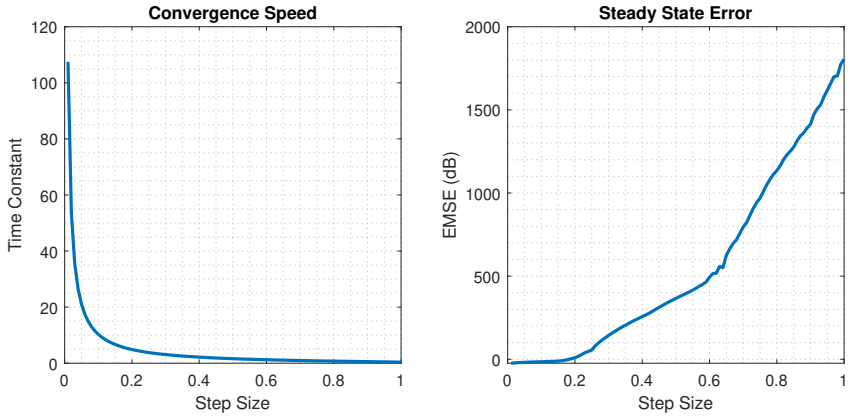


Figure 17: Graphs of steady-state error and convergence speed with step size.

.

Figure 17 illustrates the relationship between convergence speed and steady-state EMSE across different step sizes for the AR(2) process as outlined in Equation 2.1. Notably, a step size of $\mu = 0.1$ seems to be an optimal balance between convergence time and steady-state error; however, one's perspective may vary depending on the requirements of an application.

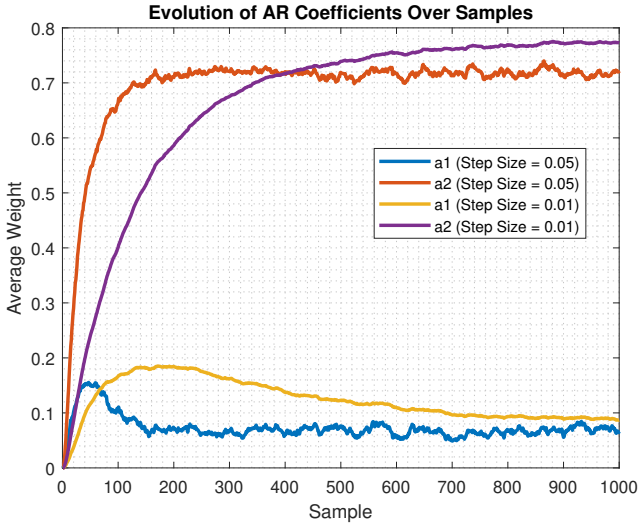


Figure 18: Evolution of AR coefficients for AR(2) process at different step sizes

Figure 18 presents a visualisation of the mean weights resulting from 100 iterations of the LMS algorithm. Notably, the parameters $a_1 = 0.1$ and $a_2 = 0.8$ were utilised. It is evident that employing a larger step size of 0.05 accelerates convergence; however, this enhancement comes at the expense of increased steady-state error. Additionally, it is noteworthy that the transient behaviour of the estimates for a_1 exhibits significant overshoot before stabilising at the steady-state.

To derive the LMS coefficient update $\mathbf{w}(n)$ minimising the given cost function, we start by expressing the cost function $J(n)$ as:

$$J(n) = \frac{1}{2}(e^2(n) + \gamma\|\mathbf{w}(n)\|_2^2) \tag{2.13}$$

where γ represents the leakage coefficient.

Firstly, we rewrite the quadratic term and 2-norm as matrix products (where $y(n)$ is the n^{th} sample of the desired output):

$$J(n) = \frac{1}{2}\left((y(n) - \mathbf{w}^{\mathbf{T}}(n)\mathbf{x}(n))(y(n) - \mathbf{w}^{\mathbf{T}}(n)\mathbf{x}(n))^T + \gamma \mathbf{w}^{\mathbf{T}}(n)\mathbf{w}(n)\right) \tag{2.14}$$

Next, we compute the gradient function with respect to $\mathbf{w}(n)$ to obtain

$$\frac{dJ(n)}{dw(n)} = -e(n)\mathbf{x}(n) + \gamma \mathbf{w}(n) \tag{2.15}$$

This gradient allows us to derive the LMS weight update algorithm. Applying the steepest descent rule, we obtain

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \mu\left(\frac{dJ(n)}{dw(n)}\right) \\ &= \mathbf{w}(n) + \mu(e(n)\mathbf{x}(n) - \gamma \mathbf{w}(n)) \\ &= (1 - \mu\gamma)\mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \end{aligned} \tag{2.16}$$

The final expression is the equation for the leaky LMS algorithm. The equation effectively stabilises the LMS algorithm by introducing a leakage coefficient that forces the filter weights to converge. Thus, the leaky LMS algorithm addresses potential instability issues encountered when the input signal $\mathbf{x}(n)$ has an autocorrelation matrix with zero eigenvalues.

The behaviour of the leaky LMS algorithm in estimating AR coefficients is influenced by several factors, including the leakage coefficient and the modification of the autocorrelation matrix. While the leaky LMS algorithm aims to ensure the invertibility of the autocorrelation matrix, it introduces a bias that affects parameter estimation reliability.

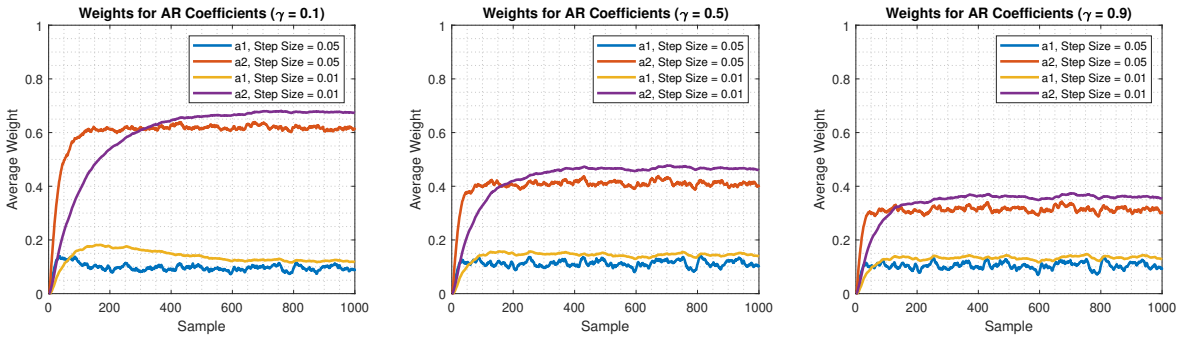


Figure 19: LMS weight evolution for AR(2) process with varying step sizes and leak coefficients

As illustrated in Figure 19, increasing the leakage coefficient results in a less smooth steady-state and weights that converge further away from the correct values. Mathematically, the leaky LMS weight update equation can be expressed as $\mathbf{w}_{\text{leaky}} = (\mathbf{R} + \gamma\mathbf{I})^{-1}\mathbf{p}$, where $\mathbf{w}_{\text{leaky}}$ represents the updated weights, $\mathbf{R}$ is the autocorrelation matrix, $\mathbf{I}$ is the identity matrix, and $\mathbf{p}$ is the cross-correlation vector between the ideal output and the input.

By ensuring the invertibility of the autocorrelation matrix, the leaky LMS algorithm introduces a bias into the parameter estimates. This bias arises due to the modification of the autocorrelation matrix to $(\mathbf{R} + \gamma\mathbf{I})$, where γ is the leakage coefficient. While this modification resolves the issue of a singular autocorrelation matrix, it can lead to parameter estimates that deviate from their true values. Consequently, the reliability of the parameter estimates may be compromised.

In summary, while the leaky LMS algorithm addresses the challenge of a singular autocorrelation matrix, its

convergence behaviour and parameter estimation accuracy are influenced by the chosen leakage coefficient. Striking a balance between stability and accuracy remains essential in the application of leaky LMS algorithms for parameter estimation.

2.2 Adaptive Step Sizes

In Section 2.1, it was evident that selecting an appropriate step size involves navigating a trade-off between convergence speed and the variance of steady-state error. Ideally, the initial step size should be relatively large and subsequently decrease as the steady-state is approached. The Gradient Adaptive Step Size (GASS) algorithms propose a method to adjust the learning rate dynamically. Within GASS, the learning rate $\mu(n)$ is updated according to:

$$\mu(n+1) = \mu(n) + \rho e(n)\mathbf{x}^{\mathbf{T}}(n)\psi(n) \tag{2.17}$$

where the term $\psi(n)$ can take one of the following forms:

- Benveniste:
$$\psi(n) = \left(\mathbf{I} - \mu(n-1)\mathbf{x}(n-1)\mathbf{x}^{\mathbf{T}}(n-1)\right)\psi(n-1) + e(n-1)\mathbf{x}(n-1) \tag{2.18}$$
- Ang & Farhang:
$$\psi(n) = \alpha\psi(n-1) + e(n-1)\mathbf{x}(n-1), \quad 0 < \alpha < 1 \tag{2.19}$$
- Matthews & Xie:
$$\psi(n) = e(n-1)\mathbf{x}(n-1) \tag{2.20}$$

These algorithms were applied on the following MA(1) process:

$$x(n) = 0.9\eta(n-1) + \eta(n), \quad \eta \sim N(0, 0.5) \tag{2.21}$$

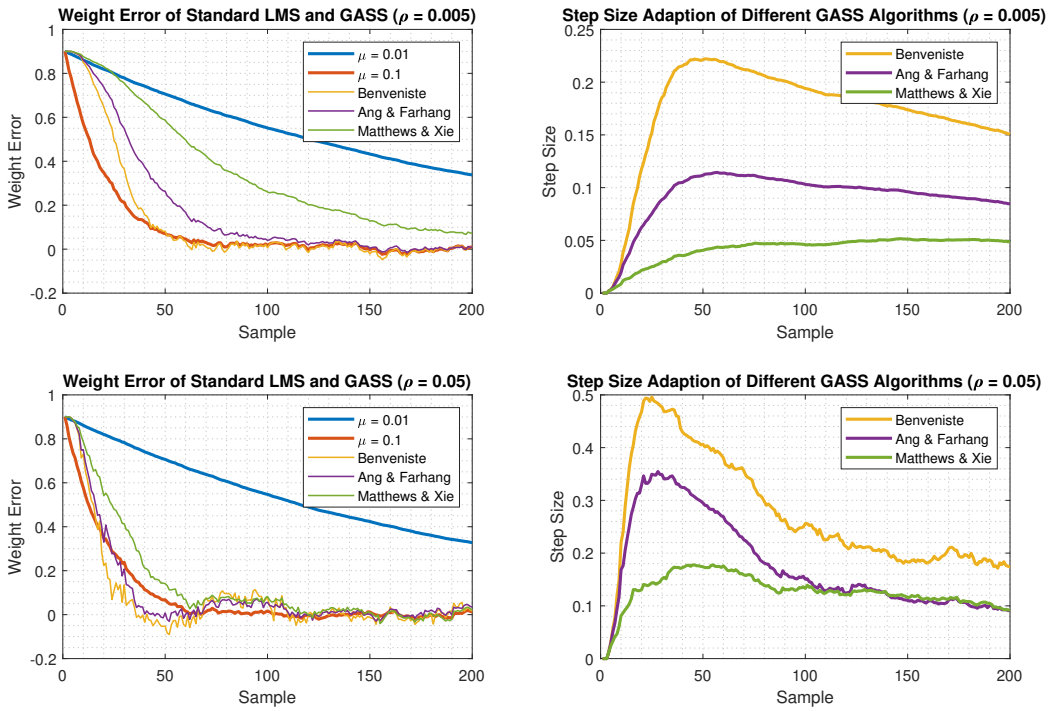


Figure 20: Transient response of standard LMS and GASS algorithms

Figure 20 depicts a comparative analysis of the transient response exhibited by various Gradient Adaptive Step Size (GASS) algorithms with initial step size $\mu^0_{GASS} = [\mu^0_B, \mu^0_{AF}, \mu^0_{MX}] = [0, 0, 0]$, in contrast to the standard Least Means Square (LMS) algorithm with step sizes $[\mu_1, \mu_2] = [0.01, 0.1]$. As anticipated, the LMS algorithm with a the smaller step size μ_1 has a slower convergence speed when compared to the larger step size μ_2.

It is noteworthy that the initial step sizes for all the GASS algorithms have been set to zero; however, both Ang & Farhang and Benveniste algorithms reach a maximum step size exceeding 0.1, so by appropriately modifying the initial step sizes, the convergence times of the GASS algorithms are likely to be significantly enhanced. Moreover, the graph indicates that a larger ρ value leads to a more rapid ascent in the step size of the GASS algorithm, thereby expediting their convergence.

Notably, of the GASS algorithms, for the given initial step sizes and ρ values, the Benveniste algorithm converges fastest, followed by the Ang & Farhang algorithm and then the Matthews & Xie algorithm. The Benveniste algorithm had the fastest convergence rate because it was most adept at scaling the step size rapidly. Conversely, the Matthews & Xie algorithm was least adept at scaling the step size rapidly.

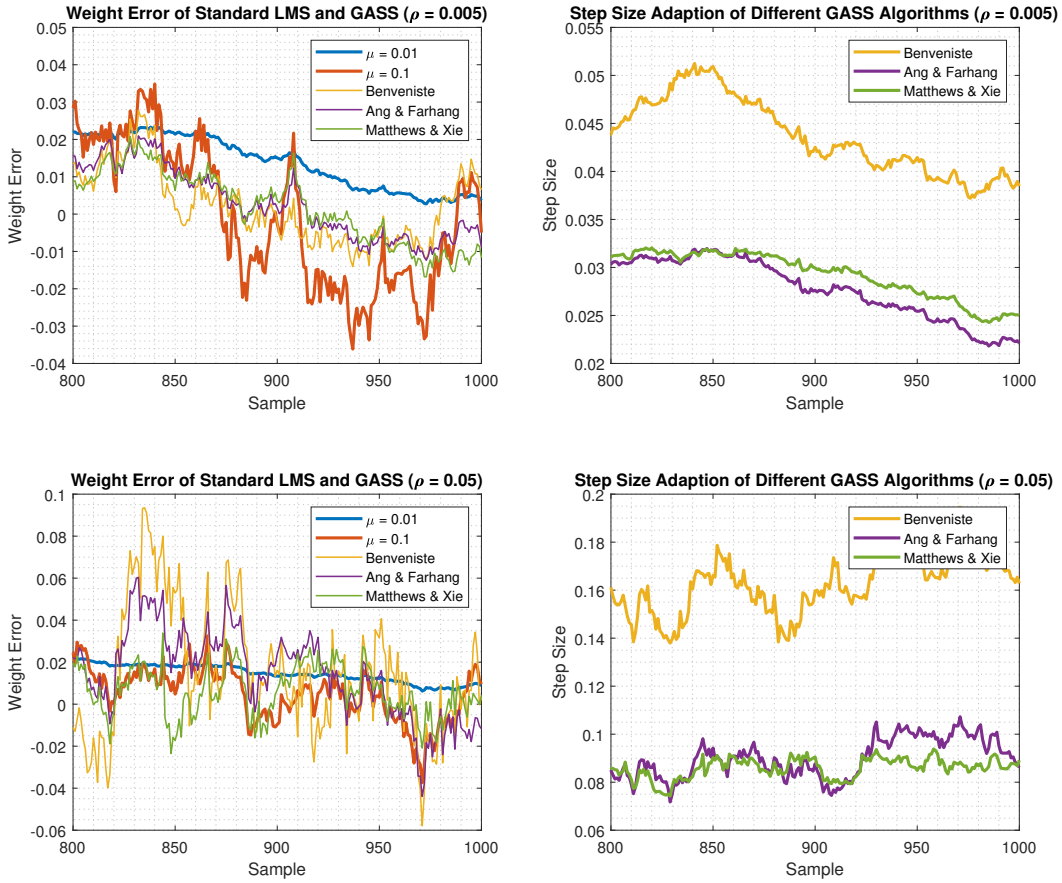


Figure 21: Steady-state response of standard LMS and GASS algorithms

Whilst Figure 20 conveys the transient response, Figure 21 depicts the complementary steady-state response. Notably, the LMS algorithm with smaller step size μ_1 demonstrates the least variance in steady-state.

When the ρ value is small (0.005), all three GASS algorithms exhibit superior performance in steady-state when compared to the LMS algorithm with step size μ_2. Conversely, when the ρ value is large (0.05), the Benveniste algorithm emerges as the least favourite performer. This observation aligns with the fact that the Benveniste

algorithm possesses the largest steady-state step size, whilst the Ang & Farhang algorithm consistently maintains the lowest step size at steady-state across both ρ values.

It is evident that the performance of the GASS algorithms is notably influenced by both the values for μ^0_{GASS} and the value for ρ. Through the optimisation of these parameters, exemplified in Figure 22 with $\rho = 0.005$ and $\mu^0_{GASS} = [0.5, 0.5, 0.5]$, the Benveniste algorithm exhibits superior adaptability characterised by steeper gradient descent and the smallest overall step size. This outcome is consistent with expectations, given that the Benveniste algorithm has the highest time complexity of $O(N^2)$, where N denotes the model order.

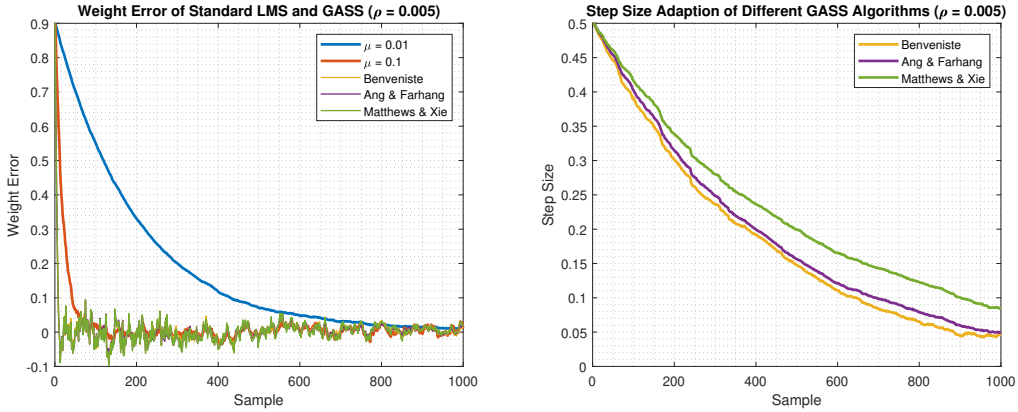


Figure 22: Complete response of standard LMS and GASS algorithms using more optimal GASS parameters

In order to verify that the update equation

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \tag{2.22}$$

based upon the *a posteriori* error $e_p(n) = d(n) - \mathbf{x}^\mathbf{T}(n)\mathbf{w}(n+1)$ is equivalent to the NLMS algorithm, we begin by substituting the update equation into the *a posteriori* error equation to obtain the *a posteriori* error in terms of the *a priori* error as follows:

$$e_p(n) = d(n) - \mathbf{x}^\mathbf{T}(n)\mathbf{w}(n+1) \tag{2.23}$$

$$e_p(n) = d(n) - \mathbf{x}^\mathbf{T}(n)(\mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n)) \tag{2.24}$$

$$e_p(n)(1 + \mu\mathbf{x}^\mathbf{T}(n)\mathbf{x}(n)) = d(n) - \mathbf{x}^\mathbf{T}(n)\mathbf{w}(n) \tag{2.25}$$

$$e_p(n)(1 + \mu\|\mathbf{x}(n)\|_2^2) = e(n) \tag{2.26}$$

$$e_p = \frac{e(n)}{(1 + \mu\|\mathbf{x}(n)\|_2^2)} \tag{2.27}$$

By substituting the equation obtained for the *a posteriori* error into the update equation, we obtain

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mu}{1 + \mu\mathbf{x}^\mathbf{T}(n)\mathbf{x}(n)}\mathbf{e}(n)\mathbf{x}(n) \tag{2.28}$$

$$= \mathbf{w}(n) + \frac{1}{\frac{1}{\mu} + \mathbf{x}^\mathbf{T}(n)\mathbf{x}(n)}\mathbf{e}(n)\mathbf{x}(n) \tag{2.29}$$

$$= \mathbf{w}(n) + \frac{\beta}{\epsilon + \mathbf{x}^\mathbf{T}(n)\mathbf{x}(n)}\mathbf{e}(n)\mathbf{x}(n) \tag{2.30}$$

which is equivalent to the NLMS algorithm for $\beta = 1$ and $\epsilon = \frac{1}{\mu}$.

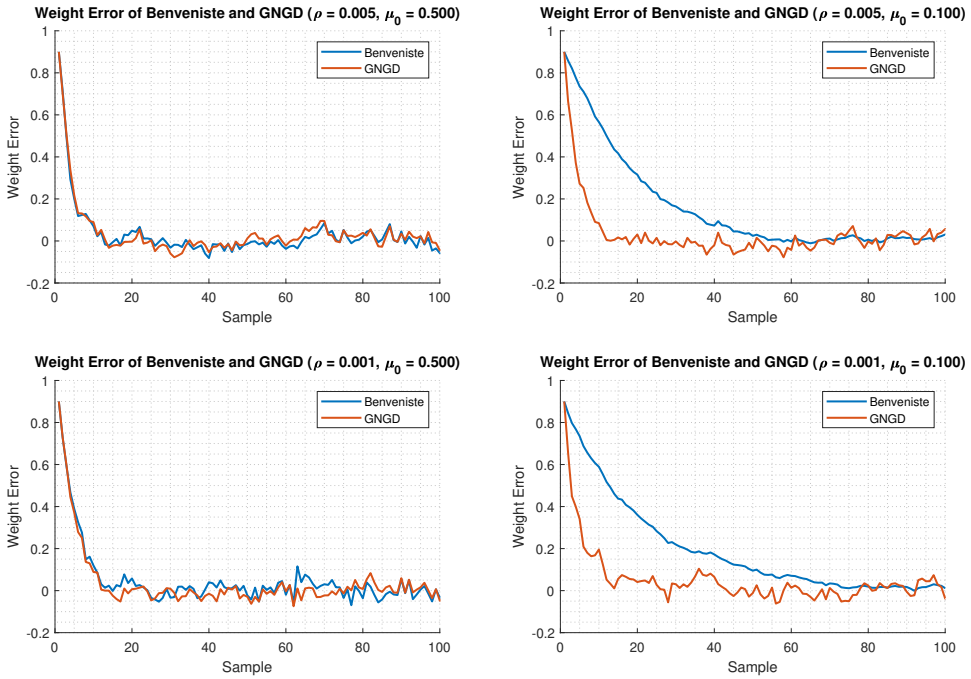


Figure 23: Weight error curves for Benveniste and GNGD algorithms using different μ_0 and ρ values

Figure 23 illustrates the weight error curves obtained for both the Benveniste GASS algorithm and the GNGD algorithm, using various combinations of μ_0 and ρ. In the plots on the right, where smaller initial step sizes are employed, the GNGD algorithm demonstrates faster convergence compared to the Benveniste algorithm. Conversely, in the plots on the left, where larger initial step sizes are used, both algorithms exhibit comparable performance.

The objective of this analysis was to emphasise the disparity in computational complexity between the two algorithms. Assuming $\mathbf{w}$ has M elements, the GNGD algorithm, with its linear computational complexity of $O(M)$, demonstrates superior scalability compared to the Benveniste GASS algorithm. The latter, characterised by a quadratic computational complexity of $O(M^2)$, is hindered by the low-pass filtering mechanism present in the update equation (2.18), which processes the instantaneous gradient.

2.3 Adaptive Noise Cancellation

In this section, we study noise suppression for two de-noising filtering configurations, utilising the noise-corrupted signal described by Equation (2.31). The fundamental signal $x(n)$ has a frequency $f_x = 0.005$ Hz, while the noise component is defined as $\eta(n) = v(n) + 0.5v(n-2)$, where $v(n)$ represents white noise with unit variance.

$$s(n) = x(n) + \eta(n) = \sin(0.01\pi n) + \eta(n) \tag{2.31}$$

The Adaptive Line Enhancer (ALE) is a noise reduction algorithm comprising a delay operator and a linear predictor. The delay parameter is meticulously selected to ensure the uncorrelation between the noise within the signal and the input to the linear predictor. The MSE can be delineated as follows:

$$\begin{aligned} E\{[s(n) - \hat{x}]^2\} &= E\{[x(n) + \eta(n) - \hat{x}(n)]^2\} && (2.32)\\ &= E\{[x(n) - \hat{x}(n)]^2\} + E\{\eta^2(n)\} + 2E\{[x(n) - \hat{x}(n)]\eta(n)\} && (2.33)\\ &= E\{[x(n) - \hat{x}(n)]^2\} + E\{\eta^2(n)\} + 2E\{x(n)\eta(n)\} - 2E\{\hat{x}(n)\eta(n)\} && (2.34) \end{aligned}$$

The term $E\{[x(n) - \hat{x}(n)]^2\}$ corresponds to the true Mean Square Prediction Error (MSPE), hence it is irrelevant to the noise for a fixed model. The second term $E\{\eta^2(n)\}$ signifies the constant noise power. Lastly, the third

term $2E\{x(n)\eta(n)\}$ equals zero due to the uncorrelated nature of the signal and noise, coupled with the zero-mean property of the noise. Therefore, only the last term, $-2E\{\hat{x}(n)\eta(n)\}$ is crucial in the minimisation of the MSE. For a given ALE system, $\hat{x}(n)$ is given by Equation (2.35), where $u(n)$ is given by equation (2.36) and M refers to the filter length.

$$\hat{x}(n) = \mathbf{w^T}(n)\mathbf{u}(n) \tag{2.35}$$

$$\mathbf{u}(n) = [s(n-\Delta), s(n-\Delta-1), \ldots, s(n-\Delta-M+1)]^T \tag{2.36}$$

We can thus delve deeper into the expression $-2E\{\hat{x}(n)\eta(n)\}$:

$$-2E\{\hat{x}(n)\eta(n)\} = -2\mathbf{w^T}\begin{pmatrix} E\{\eta(n)(x(n-\Delta)-\eta(n-\Delta))\} \\ \vdots \\ E\{\eta(n)(x(n-\Delta-M+1)-\eta(n-\Delta-M+1))\} \end{pmatrix} \tag{2.37}$$

In the expansion of each term, as before, considering the deterministic signal and the zero-mean noise term, the first term vanishes. Furthermore, the negative terms cancel each other out, resulting in a vector comprising elements from the ACF of $\eta(n)$. Given the expression, $E\{\eta(n)\eta(n-k)\} = 2\delta(k)+0.5\delta(k-2)$ for this model, and considering $M > 1$, we determine that $\Delta_{\min} = 3$, is the minimum Δ to set the above term to 0. This is expected as $\eta(n)$ is an MA(2) process.

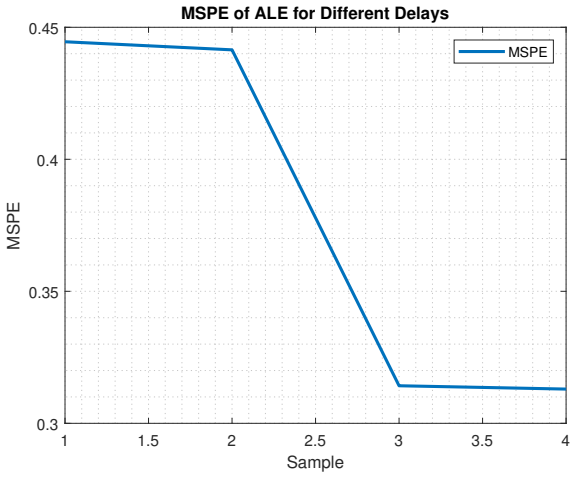


Figure 24: MSPE for different delays

Figure 24 clearly reinforces this conclusion as the MSPE reaches its minimum value for $\Delta \geq 3$.

Figure 25 (left) illustrates the relationship between ALE MSPE and delay for a range of filter orders. According to the graph, the best choice of parameters are $M = 5$ and $\Delta = 3$. The right-hand subplot illustrates the efficacy of these parameter choices. Clearly, there is an improvement in SNR.

A large model order leads to excess degrees of freedom, thereby amplifying computational complexity whilst diminishing model efficacy. This is due to the model's propensity to overfit the noise component as an integral part of the signal. We observe this phenomenon in the left-hand plot in Figure 25. Moreover, for $\Delta > 5$, the performance of the ALE model deteriorates progressively as indicated by the escalating MSPE. These results reflect the need for determining the appropriate hyperparameters for a given task.

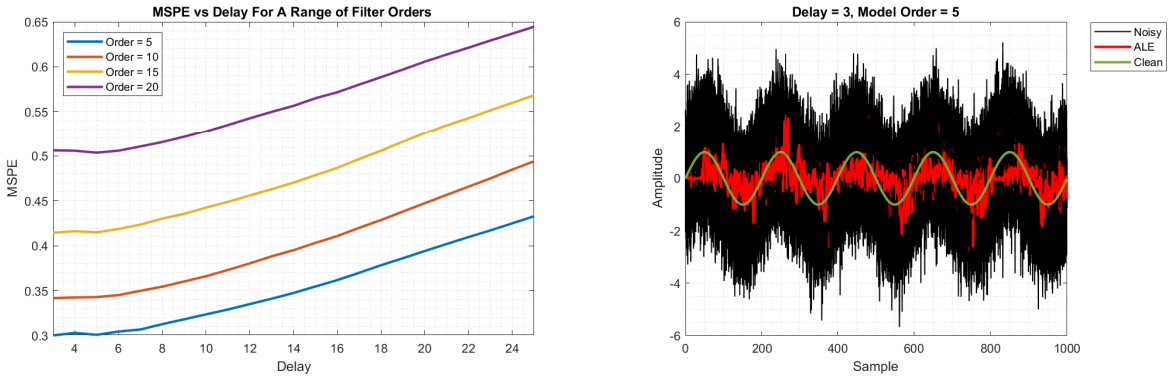


Figure 25: ALE performance for different delays and filter orders

The ANC algorithm works by taking a noisy signal (2.31), along with an additional noise input that exhibits some level of correlation with the primary noise. The secondary noise is passed to a linear predictor which aims to learn the primary noise component. This primary noise estimation is then subtracted from the original noisy input signal to yield the denoised signal.

Figure 26 illustrates the efficacy of both the ANC and ALE algorithms for the same noisy input signal. Notably, the ANC output matches the clean signal more closely when compared to the ALE output. This assertion is supported by the MSPE values, with $MSPE_{ANC}$ recorded at 0.1084 and $MSPE_{ALE}$ at 0.3137. Based on this statistic, the ANC algorithm can be said to perform 65.4% better than the ALE algorithm.

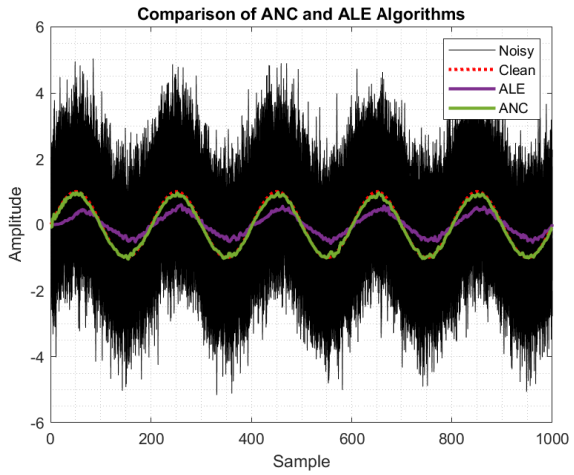


Figure 26: Performance comparison of ANC and ALE algorithms

Figure 27 illustrates the efficacy of the ANC filtering configuration in eliminating the 50Hz mains interference from a real dataset for a range of ANC parameters. From the subplots, it appears that both filters with step size $\mu = 0.001$ and model order $M = [10, 15]$ are most effective at removing the 50Hz component, whilst preserving the signal component; however, the filter with model order 10 is chosen due to its reduced computational complexity when compared to the model order of 15. This choice yields an MSPE value of 1.2022×10^{-11}, indicating a satisfactory level of denoising performance.

However, it's crucial to recognise that MSPE alone may not provide a comprehensive assessment of denoising effectiveness. Other factors, such as the nature of the original signal and the desired level of interference removal, should also be considered. Consequently, while the ANC algorithm demonstrates significant potential in noise reduction applications, its optimal configuration may vary depending on specific requirements.

In practical scenarios, ANC algorithms find wide-ranging applications, notably in premium noise-cancelling head-

phones such as the Sony WH-1000XM5 and Airpod Max. These real-world implementations exemplify the practical relevance and effectiveness of ANC techniques in mitigating environmental noise for enhanced user experience.

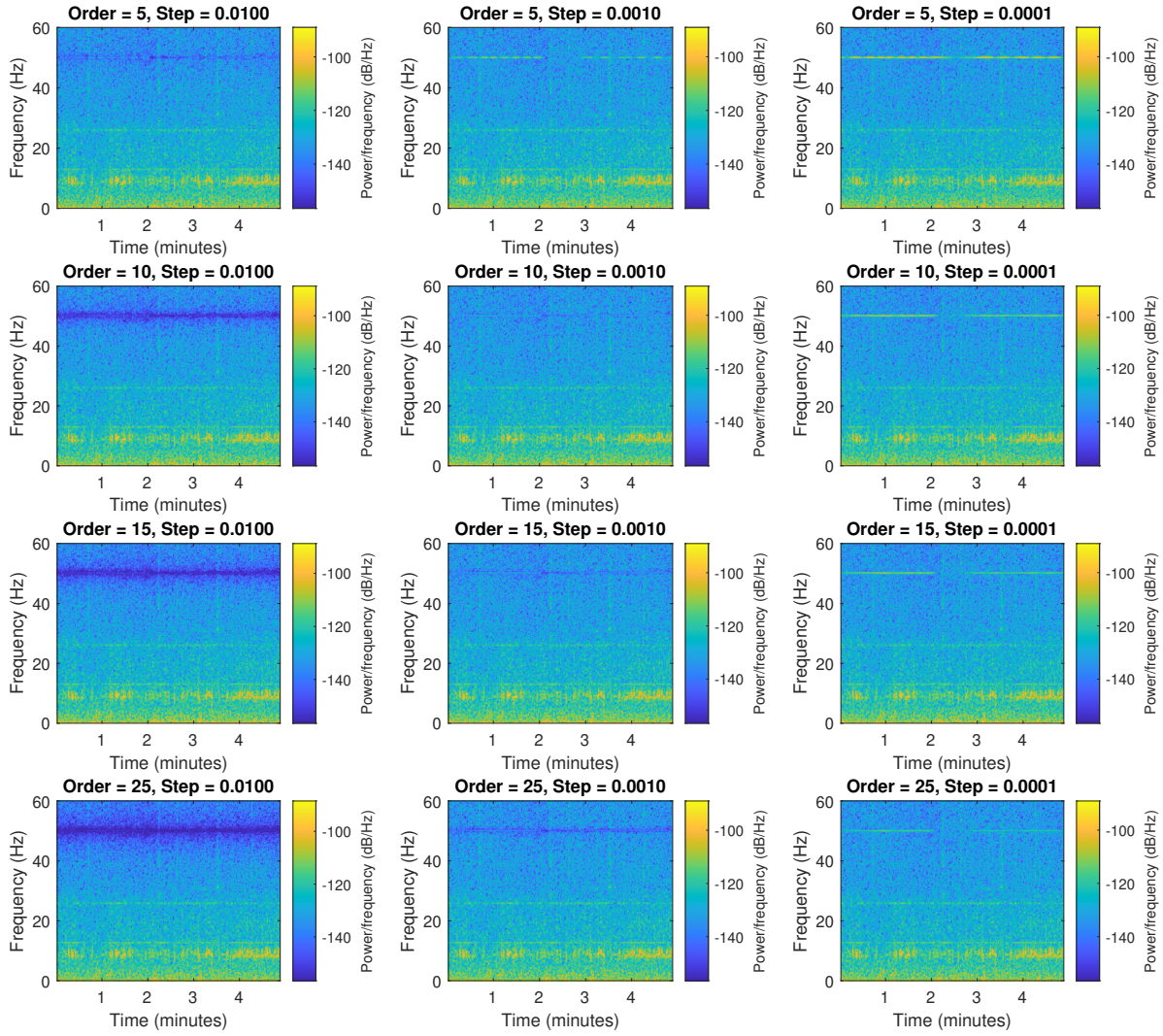


Figure 27: Spectrograms of ANC cleaned signal for different ANC parameters

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

In this section, we investigate the use of complex-valued signals and evaluate the performance of CLMS and ACLMS algorithms in processing such signals for system identification and prediction tasks. We begin by generating a first-order widely-linear-moving-average process, WLMA(1), driven by circular white noise, $x(n)$

$$y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-1), \quad x \sim \mathcal{N}(0,1) \tag{3.1}$$

where $b_1 = 1.5 + 1j$ and $b_2 = 2.5 - 0.5j$. Figure 28 illustrates the distribution of both signals, $x(n)$ and $y(n)$. As expected, the distribution of the WGN is more circular in comparison to the WLMA(1) process.

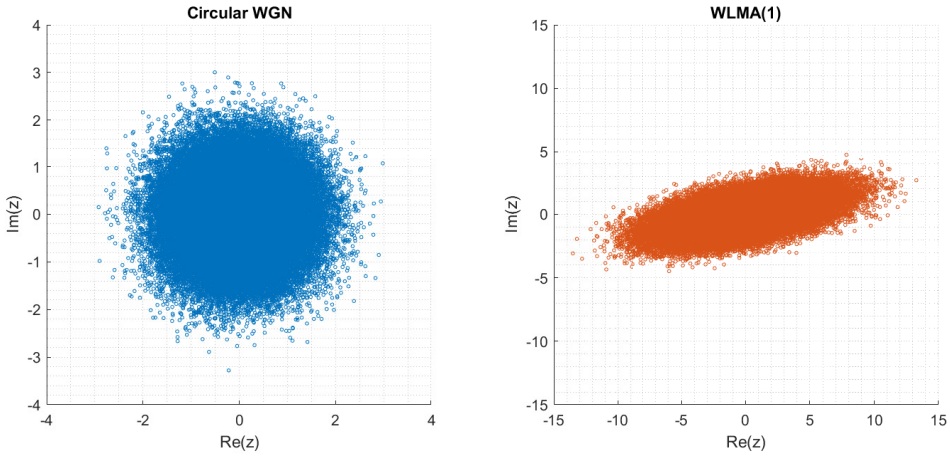


Figure 28: Circularity of circular WGN (left) and WLMA(1) (right)

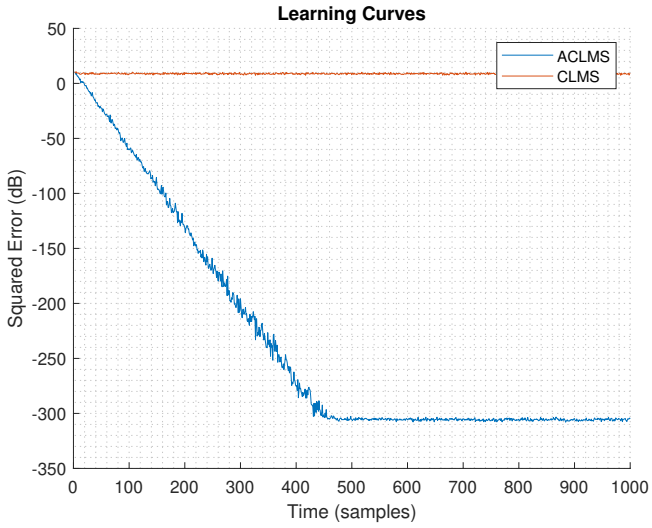


Figure 29: Learning curves of circular WGN and WLMA(1)

The efficacy of both the CLMS and ACLMS algorithms in estimating the generated WLMA(1) process is then examined. Figure 29 shows the learning curves for both algorithms. As expected, the ACLMS algorithm performs magnitudes better at correctly estimating b_1 and b_2 than the standard CLMS algorithm, since the CLMS algorithm only caters to circular data. Without the presence of a conjugate factor, CLMS is unable to correctly identify the coefficients, settling at a steady-state error of approximately 7.5dB. On the other hand, the ACLMS algorithm settles at a steady-state error of approximately -307.5dB, indicating that the coefficients b_1 and b_2

have been estimated correctly. From this, we can clearly see the limitation of the CLMS algorithm in estimating coefficients when the signal is non-circular. In contrast, the enhanced capabilities of the ACLMS enable it to effectively handle such situations and achieve accurate estimation. The only case in which the CLMS algorithm would be a wiser choice over the ACLMS algorithm is if a faster convergence time is required to estimate circular data, attributable to its utilisation of half the parameters required for training, compared to the ACLMS algorithm.

We proceed by loading the bivariate wind data and constructing the complex signal $v[n] = v_{\text{east}}[n] + jv_{\text{north}}[n]$ for the three wind regimes: low, medium, and high. Figure 30 shows the circularity plots for the three regimes on the left-hand side with corresponding plots of MSPE against filter order, adjacent to them, on the right-hand side.

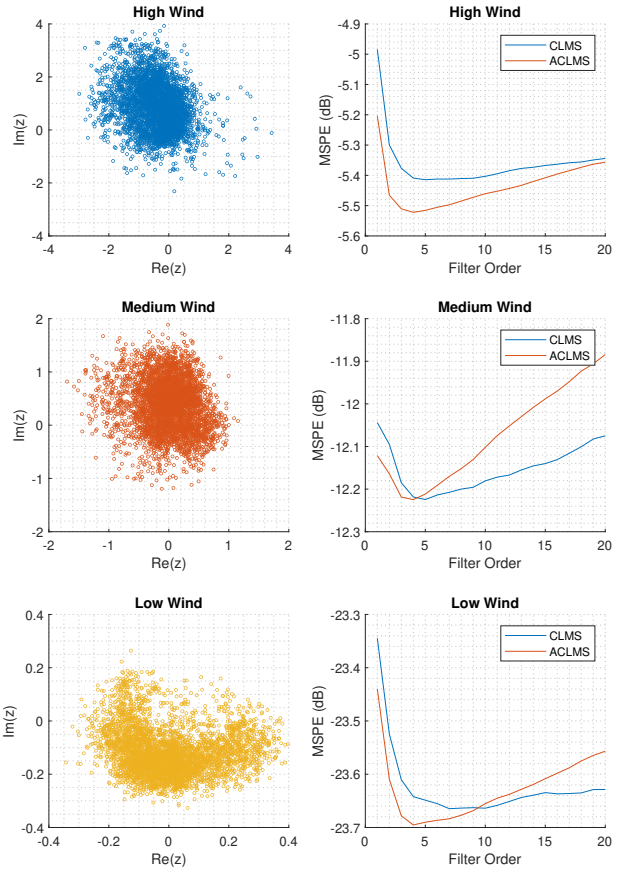


Figure 30: Circularity plots of wind data for three regimes, along with corresponding plots of MSPE vs Filter Order

The circularity coefficient, ρ, measures how rotation-invariant the distribution of a signal is about the origin, so it's a standard measure used to quantify the circularity of a signal. A value closer to 0 indicates strong circularity, whilst values closer to 1 indicate weak circularity. The circularity coefficients were determined to be $\rho_{high} = 0.623$, $\rho_{medium} = 0.454$ and $\rho_{low} = 0.159$. As anticipated, high wind conditions exhibit the greatest variability, resulting in the lowest circularity. Conversely, low wind conditions demonstrate the least variation and consequently have the highest circularity. Medium wind conditions intuitively falls between both of these extremes.

The CLMS and ACLMS filters were employed in a predictive context to conduct one-step-ahead predictions of the complex wind data for each regime. With the aim of minimising prediction errors while ensuring filter stability, the optimal learning rates were determined to be $\mu_{high} = 0.001$, $\mu_{medium} = 0.01$ and $\mu_{low} = 0.1$. A grid-search was conducted for all model orders from 1 to 20 and the right-hand plots in Figure 30 depict the MSPE for different

filters and regimes. The results indicate that the ACLMS algorithm performs better than CLMS algorithm if at the optimal model order. From the graphs, the optimal model order for each regime has been determined to be $M = 4$. Notably, the ACLMS algorithm performs better than the CLMS algorithm for $M \leq 4$, but for higher model orders, given the larger number of degrees of freedom of ACLMS relative to CLMS, overfitting increases more rapidly as a function of model order.

The complex-valued voltage representation of a three-phase system can be derived from the Clarke Transform as

$$v(n) = A(n)e^{j\left(2\pi\frac{f_0}{f_s}n+\phi\right)} + B(n)e^{-j\left(2\pi\frac{f_0}{f_s}n+\phi\right)} \tag{3.2}$$

where

$$A(n) = \frac{\sqrt{6}}{6}\left(V_a(n) + V_b(n)e^{j\Delta_b} + V_c(n)e^{j\Delta_c}\right) \tag{3.3}$$

and

$$B(n) = \frac{\sqrt{6}}{6}\left(V_a(n) + V_b(n)e^{-j(\Delta_b+\frac{2\pi}{3})} + V_c(n)e^{-j(\Delta_c-\frac{2\pi}{3})}\right) \tag{3.4}$$

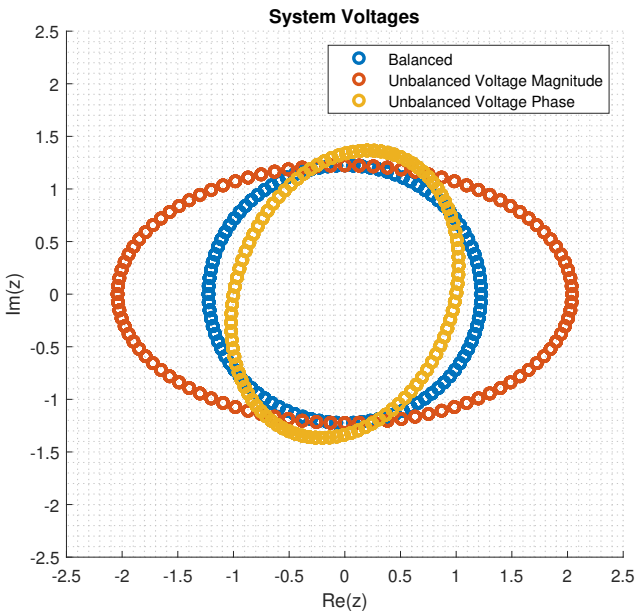


Figure 31: Circularity plots for balanced and unbalanced systems

The system is considered balanced when $V_a(n) = V_b(n) = V_c(n)$ and $\Delta_b = \Delta_c = 0$. In this state, the system exhibits complete circular behaviour, demonstrating rotational invariance. However, when the voltage magnitudes or voltage phase shifts are altered, causing the system to become unbalanced, the system exhibits deviation from complete circularity. Figure 31 illustrates these described phenomena. Thus, by analysing the circularity and the extent of deviation, the circularity diagram serves as a valuable tool for detecting faults within the system.

The frequency of a balanced and unbalanced system can be derived from the coefficients learned from the first order Strictly Linear Autoregressive (SLAR(1)) and first order Widely Linear and Autoregressive (WLAR(1)) models:

$$\text{Strictly Linear}: \; v(n+1) = h^*(n)v(n) \tag{3.5}$$

$$\text{Widely Linear}: \; v(n+1) = h^*(n)v(n) + g^*(n)v^*(n) \tag{3.6}$$

Under balanced conditions, the complex voltage can be expressed as

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \tag{3.7}$$

Hence, via the substitution of (3.7) into (3.5), we obtain

$$v(n+1) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} = h^*(n) \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \tag{3.8}$$

and we deduce $h^*(n)$ to be equal to $e^{j(2\pi \frac{f_0}{f_s})}$. In order to write the LHS in terms of real and imaginary components, we realise that it can be rewritten as

$$h^*(n) = e^{-j \arctan(\frac{\Im(h(n))}{\Re(h(n))}} \tag{3.9}$$

from which we can arrive at

$$2\pi \frac{f_0}{f_s} = -\arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right) \tag{3.10}$$

$$f_0 = -\frac{f_s}{2\pi} \arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right) \tag{3.11}$$

Since the negative sign only determines the direction of rotation, it can be neglected and we arrive at our desired result:

$$f_0 = \frac{f_s}{2\pi} \arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right) \tag{3.12}$$

The same reasoning can be applied for the unbalanced system, whereby substituting (3.3) into (3.6) results in

$$\begin{aligned} &A(n+1)e^{j\left(2\pi \frac{f_0}{f_s}(n+1)+\phi\right)} + B(n+1)e^{-j\left(2\pi \frac{f_0}{f_s}(n+1+\phi\right)} \\ &= h^*(n)(A(n)e^{j\left(2\pi \frac{f_0}{f_s} n+\phi\right)} + B(n)e^{-j\left(2\pi \frac{f_0}{f_s} n+\phi\right)}) \\ &\quad + g^*(n)(A^*(n)e^{-j\left(2\pi \frac{f_0}{f_s} n+\phi\right)} + B^*(n)e^{j\left(2\pi \frac{f_0}{f_s} n+\phi\right)}) \end{aligned} \tag{3.13}$$

Comparing the coefficients of $e^{j\left(2\pi \frac{f_0}{f_s} n+\phi\right)}$ and $e^{-j\left(2\pi \frac{f_0}{f_s} n+\phi\right)}$, we get

$$A(n+1)e^{j(2\pi \frac{f_0}{f_s})} = h^*(n)A(n) + g^*(n)B^*(n) \tag{3.14}$$

$$B(n+1)e^{-j(2\pi \frac{f_0}{f_s})} = h^*(n)B(n) + g^*(n)A^*(n) \tag{3.15}$$

If we assume $v(n+1) \approx v(n)$, we can then assume $A(n+1) = A(n)$ and $B(n+1) = B(n)$. Therefore, we can say that

$$e^{j(2\pi \frac{f_0}{f_s})} = \frac{h^*(n)A(n) + g^*(n)B^*(n)}{A(n+1)} \approx h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \tag{3.16}$$

$$e^{-j(2\pi \frac{f_0}{f_s})} = \frac{h^*(n)B(n) + g^*(n)A^*(n)}{B(n+1)} \approx h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \tag{3.17}$$

By conjugating the RHS of (3.17), we can equate with the RHS of (3.16) to get

$$h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} = h(n) + g(n)\frac{A(n)}{B^*(n)} \tag{3.18}$$

Then, by letting $z = \frac{B^*(n)}{A(n)}$, we get

$$h^*(n) + g^*(n)z = h(n) + \frac{g(n)}{z} \tag{3.19}$$

$$h^*(n)z + g^*(n)z^2 = h(n)z + g(n) \tag{3.20}$$

$$g^*(n)z^2 + (h^*(n) - h(n))z - g(n) = 0 \tag{3.21}$$

This second order equation can be solved using the quadratic formula to get

$$\frac{B^*(n)}{A(n)} = j\frac{\Im(h(n)) \pm \sqrt{\Im(h(n))^2 - |g(n)|^2}}{g^*(n)} \tag{3.22}$$

Substituting back into (3.16), we get

$$e^{j(2\pi\frac{f_0}{f_s})} = h^*(n) + g^*(n)j\frac{\Im(h(n)) \pm \sqrt{\Im(h(n))^2 - |g(n)|^2}}{g^*(n)} \tag{3.23}$$

$$e^{j(2\pi\frac{f_0}{f_s})} = \Re(h(n)) \pm j\sqrt{\Im(h(n))^2 - |g(n)|^2} \tag{3.24}$$

$$e^{j(2\pi\frac{f_0}{f_s})} = |C|e^{j\left(\arctan\frac{\pm\sqrt{\Im(h(n))^2-|g(n)|^2}}{\Re(h(n))}\right)} \tag{3.25}$$

Since $|e^{j(2\pi\frac{f_0}{f_s}}| = 1$, $|C| = 1$. Therefore, we can equate the exponents to get

$$2\pi\frac{f_0}{f_s} = \arctan\frac{\pm\sqrt{\Im(h(n))^2 - |g(n)|^2}}{\Re(h(n))} \tag{3.26}$$

$$f_0 = \pm\frac{f_s}{2\pi}\arctan\frac{\sqrt{\Im(h(n))^2 - |g(n)|^2}}{\Re(h(n))} \tag{3.27}$$

and we get the desired result by selecting the positive frequency:

$$f_0 = \frac{f_s}{2\pi}\arctan\frac{\sqrt{\Im(h(n))^2 - |g(n)|^2}}{\Re(h(n))} \tag{3.28}$$

Based on the preceding findings, we can estimate the $\alpha - \beta$ voltages from earlier using the coefficients of either a SLAR(1) or WLAR(1) model. To achieve this, we set $f_0 = 50$, $f_s = 5000$, $\mu = 0.01$, $\phi = 0$ and employ the previously defined CLMS and ACLMS algorithms for frequency estimation.

Given that we are more concerned with the steady-state value that both the CLMS and ACLMS algorithms predict for this task, the initial conditions of each algorithm haven't been considered for the following task. Figure 32 illustrates the frequency estimation graphs for the different systems. As expected, whilst both the CLMS and ACLMS algorithms are able to learn the appropriate coefficients for estimating the frequency of a balanced system accurately, due to being second-order circular, only the ACLMS algorithm converges to the correct frequency for imbalanced systems. In addition, despite the initial oscillations observed for both algorithms in the unbalanced system case, only the ACLMS algorithm is able to overcome this and converge to the correct frequency.

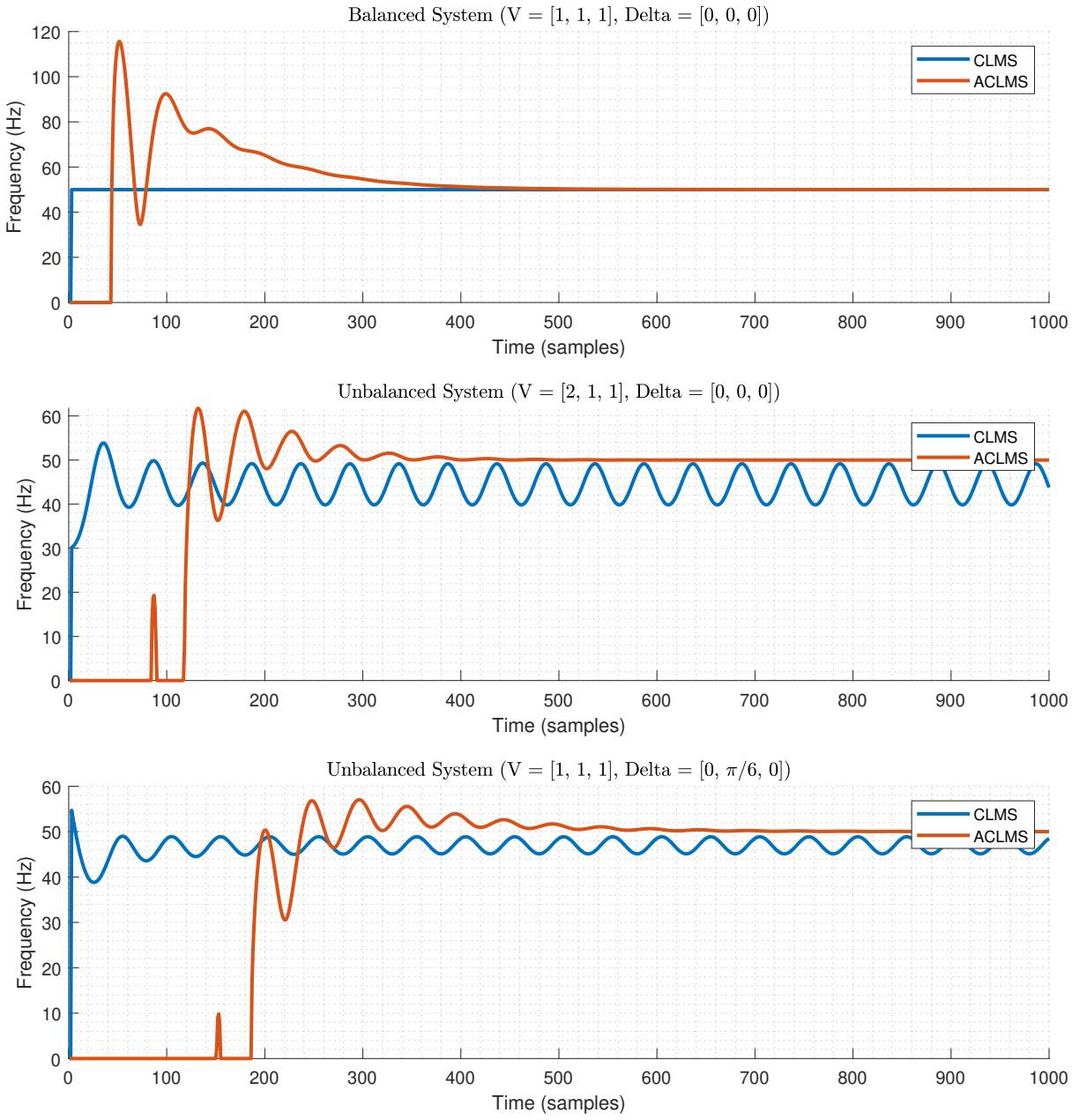


Figure 32: Frequency estimation using CLMS and ACLMS algorithms for a balanced system, and two unbalanced systems (magnitude and phase distortions)

3.2 Adaptive AR Model Based Time-Frequency Estimation

In this section, we explore the concepts of non-stationary spectrum estimation. We begin by generating the frequency modulated (FM) signal $y(n) = e^{j(\frac{2\pi}{f_s}\phi(n)} + \eta(n)$, where $\eta(n) \sim \mathcal{N}(0, 0.05)$ and $\phi(n)$ is generated from 3.29.

$$\frac{d\phi(n)}{dn} = \begin{cases} 100 & 1 \leq n \leq 500 \\ 100 + \frac{n-500}{2} & 501 \leq n \leq 1000 \\ 100 + \left(\frac{n-1000}{25}\right)^2 & 1001 \leq n \leq 1500 \end{cases} \tag{3.29}$$

Figure 33 (left) depicts the evolution of frequency for the FM signal, whilst Figure 33 (right) depicts the power spectral estimates obtained for different AR models. Autoregressive spectral estimation operates under the assumption of stationarity, which limits its ability to accurately track frequency fluctuations across different model orders. Instead, it aims to identify the most suitable stationary representation, thereby capturing the average frequency content of the signal. This behaviour is exemplified by the AR(1) peak, which corresponds to the average frequency of the FM signal, denoted as 'Mean FM Signal Freq'.

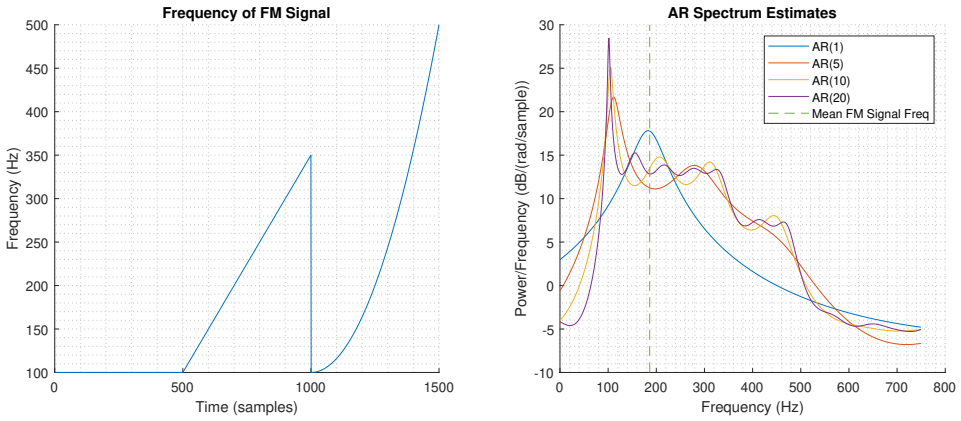


Figure 33: Frequency content of the FM signal (left) and AR spectral estimates (right)

We then implement the CLMS algorithm to estimate the AR coefficients of the signal $y(n)$ at each time instant $y(n+1) = a_1^*(n)y(n)$. After removing outliers, we obtain the time-frequency spectrum, and note that for $\mu = 0.05$, the plots resemble the frequency variation seen in Figure 33. This is illustrated in Figure 34.

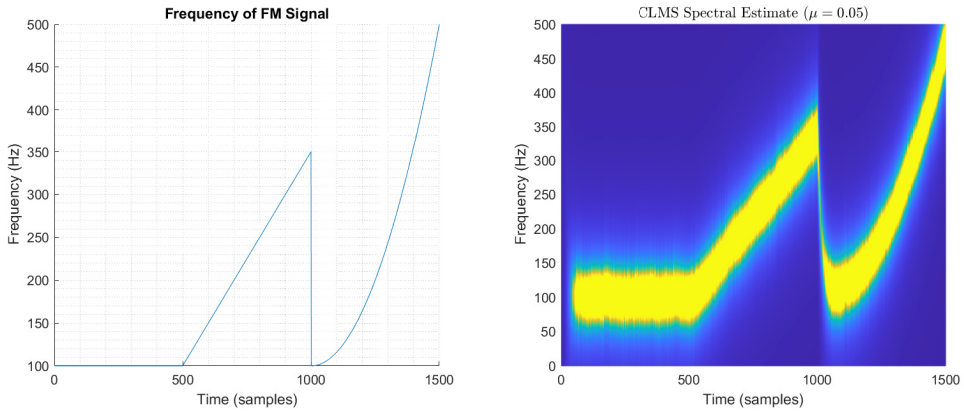


Figure 34: Frequency content of the FM signal (left) and AR spectral estimates (right)

As seen in Figure 35, by varying μ, we note that when μ is exceedingly small, such as 0.005, the learning rate becomes sluggish, leading to slow convergence. Consequently, the estimation is only available for larger values of n. We also note that the actual estimation is not sufficiently adaptive and can't replicate the bigger changes in the linear and exponential regions, compromising the final estimation. On the other hand, if μ is set too high (e.g. 0.5), the estimation becomes very noisy. Notably, a higher value of μ does capture instantaneous changes as demonstrated at $n = 1000$. Overall, a value of $\mu = 0.05$ strikes a favourable balance between the benefits and drawbacks of excessively low or high values, distinctly depicting the shape and frequency variations.

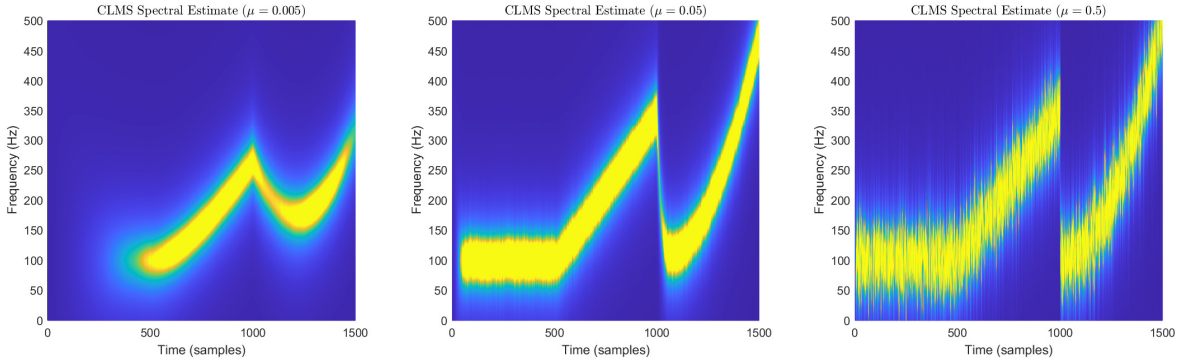


Figure 35: AR spectral estimates for different values of μ

3.3 A Real Time Spectrum Analyser Using Least Mean Square

A signal, $y(n)$, can be approximated as a linear combination of N harmonically related sinusoids, given by

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k) e^{j2\pi kn/N} \tag{3.30}$$

where $\hat{y}(n)$ is the signal estimate and $w(k)$ are the unknown weights to be estimated. These weights correspond to the Fourier coefficients obtained from the Discrete Fourier Transform (DFT) of the signal $y(n)$.

By collecting all N estimates of the signal $y(n)$ into a vector, we can express the problem in vector form as $\hat{\mathbf{y}} = \mathbf{F}\mathbf{w}$.

$$\begin{bmatrix} \hat{y}(0) \\ \hat{y}(1) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{j\frac{2\pi}{N}(1)(1)} & \cdots & e^{j\frac{2\pi}{N}(1)(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{j\frac{2\pi}{N}(N-1)(1)} & \cdots & e^{j\frac{2\pi}{N}(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ \vdots \\ w(N-1) \end{bmatrix} \tag{3.31}$$

To determine the optimal weights $\mathbf{w}$, we minimise the sum of squared errors between the estimated signal and the true signal:

$$\min_{\mathbf{w}} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \min_{\mathbf{w}} \sum_{n=0}^{N-1} |y(n) - \hat{y}(n)|^2 \tag{3.32}$$

This equation can be further simplified as

$$\min_{\mathbf{w}} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \min_{\mathbf{w}} (\mathbf{y} - \mathbf{F}\mathbf{w})^H (\mathbf{y} - \mathbf{F}\mathbf{w}) \tag{3.33}$$

$$= \min_{\mathbf{w}} (\mathbf{y}^H\mathbf{y} - \mathbf{w}^H\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}\mathbf{w} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w}) \tag{3.34}$$

Given that the two middle terms are scalars and each other's transposes, we have

$$\min_{\mathbf{w}} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \min_{\mathbf{w}} (\mathbf{y}^H\mathbf{y} - 2\mathbf{w}^H\mathbf{F}^H\mathbf{y} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w}) \tag{3.35}$$

Taking the gradient with respect to w and setting it to zero yields the equation for the optimal weights in the least-squares sense, as shown in Equation (3.36):

$$\frac{\partial \|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\partial \mathbf{w}} = 0 = -2\mathbf{F}^H\mathbf{y} + 2\mathbf{F}^H\mathbf{F}\mathbf{w} \tag{3.36}$$

This can be further simplified to obtain the optimal weights.

$$\mathbf{w} = (\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{y} \tag{3.37}$$

Considering that $\mathbf{F}$ is multiplied by $\mathbf{w}$ to obtain $\hat{\mathbf{y}}$, it follows that this estimate lies within the subspace spanned by the columns of $\mathbf{F}$, referred to as its column space $C(\mathbf{F})$. Thus, this estimate can be interpreted as the projection of $\mathbf{y}$ onto $C(\mathbf{F})$ that minimises the sum of squared errors. This interpretation is reinforced by the projection matrix, denoted as $\mathbf{P}$, which is derived by substituting (3.37) into $\hat{\mathbf{y}} = \mathbf{F}\mathbf{w}$ to obtain

$$\mathbf{P} = \mathbf{F}(\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H \tag{3.38}$$

$$\hat{\mathbf{y}} = \mathbf{P}\mathbf{y} \tag{3.39}$$

This projection matrix serves to project $\mathbf{y}$ onto the column space of $\mathbf{F}$, providing a more accurate estimate, $\hat{\mathbf{y}}$.

Similar to the approach described in Section 3.2, leveraging the DFT as a least squares solution enables the development of an adaptive implementation using the CLMS algorithm. In this adaptive version, the input to the filter at time n is defined as

$$\mathbf{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} e^{j\frac{2\pi kn}{N}} \tag{3.40}$$

Figure 36 depicts the spectrum of the original signal. The plot illustrates the algorithm's successful handling of the signal's constant period initially, followed by its consideration of the linear and exponential sections. However, a notable observation is the persistent nature of the signal components, leading to the observed overlapping plot. This persistence arises from the lack of timely updates to the Fourier weights in CLMS, a gradient-based algorithm. The absence of block-based updates results in slow error backpropagation, causing memory to fade gradually and inhibiting proper adaptation to new signal sections. This behaviour is particularly evident in the first sinusoid corresponding to the constant section, which remains significant throughout other signal segments.

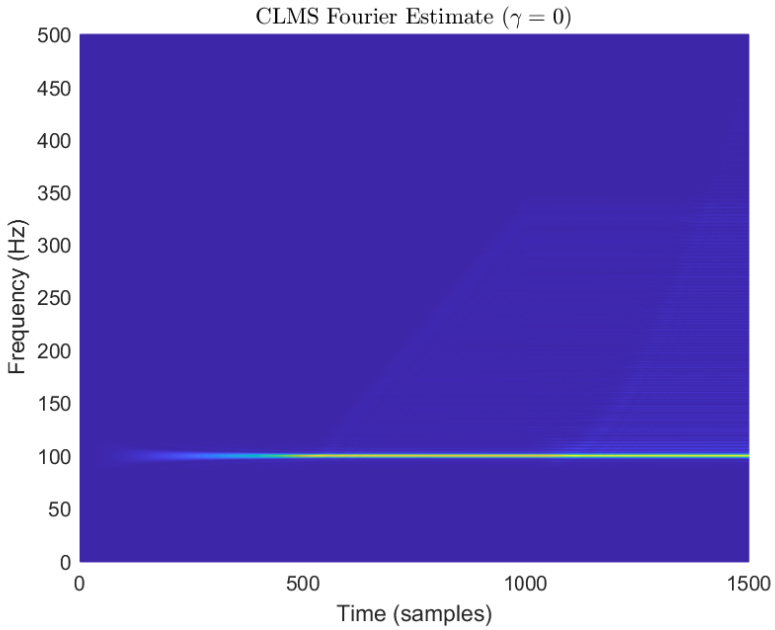


Figure 36: Time-Frequency spectrum of FM signal using the standard DFT-CLMS algorithm

To address the slow update issue encountered with the standard DFT-CLMS, a modified version of the algorithm is proposed inspired by the principles of the Leaky LMS discussed in Section 2.1. This adaptation, termed Leaky DFT-CLMS, is formulated as

$$w(n+1) = (1 - \gamma\mu)w(n) + \mu e^*(n)x(n) \tag{3.41}$$

Here, the leak parameter, γ, introduces a forgetting factor, allowing the update to prioritise recent information over past weights, thereby facilitating faster adaptation. The impact of γ is examined in Figure 37. A small γ still results in sluggish updates, failing to address the initial issue due to its limited effect. Conversely, a large γ yields a noisy output, where rapid fading renders only the most recent samples significant, resulting in a less smooth output. The optimal setting found was $\gamma = 0.05$, which strikes a balance between rapid updates and signal fidelity. The corresponding spectrum estimate exhibits a close match to the original signal's frequency variations, featuring a narrower bandwidth when compared to the previous CLMS approach.

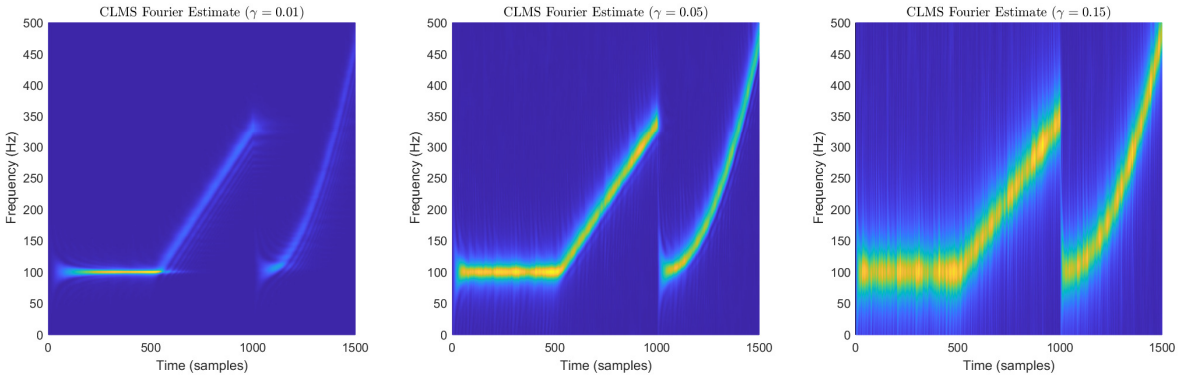


Figure 37: Time-Frequency spectrum of FM signal using the Leaky DFT-CLMS algorithm

Finally, spectrum estimation was conducted on the EEG POz dataset from Part 1.2. To reduce the computational complexity, the analysis was restricted to the range n ranging from 10000 to 11199. Given the stationary nature of the data, the use of Leaky DFT-CLMS was deemed unnecessary, with the standard DFT-CLMS algorithm alone proving adequate. Figure 38 depicts the resulting time-frequency spectrum. The SSVEP signal at $26Hz$ is distinctly discernible, alongside prominent mains interference at $50Hz$. There is also a semblance of the first and third harmonics at $13Hz$ and $39Hz$ but this is faint. Notably, the first few harmonics of the 3Hz signal have been distinctly discerned. Overall, although this spectral estimation approach yields spectral estimates with reduced noise, it entails heightened computational complexity and necessitates a substantial number of observations to learn the optimal parameters.

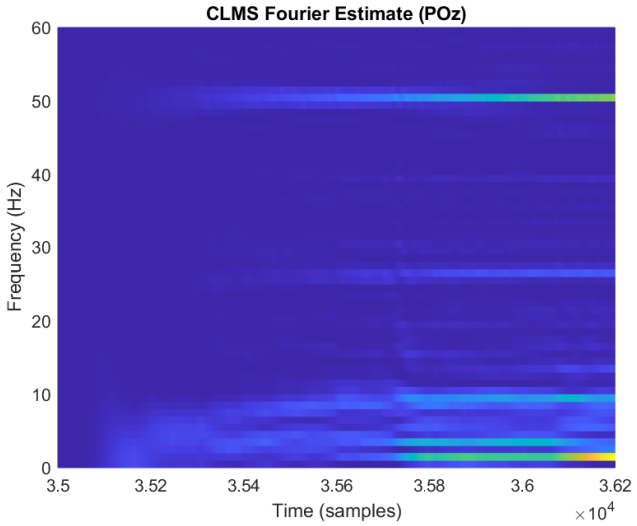


Figure 38: Time-Frequency spectrum of EEG POz signal using the standard DFT-CLMS algorithm

4 From LMS to Deep Learning

In this section, we explore further the performance of the LMS algorithm when applied to non-stationary streaming data, and show how it can be extended to address a wide range of non-linear regression problems.

We begin by loading the non-stationary, non-zero mean time-series from 'time-series.mat' and detrending the signal to make the signal zero-mean so that it is suitable for one-step-ahead prediction. To adaptively learn from the non-stationary signal, the LMS algorithm is employed (with a learning rate of $\mu = 1 \times 10^{-5}$) and assuming the data is generated using an AR(4) process. The prediction of the subsequent sample, $y[n]$, is thus based on the four preceding samples, $y[n-1]$, $y[n-2]$, $y[n-3]$ and $y[n-4]$. As seen in Figure 39, the LMS algorithm is highly inaccurate initially but it converges after approximately $200 - 300$ samples, achieving a satisfactory prediction of the signal, as shown in the zoomed plot. Despite the observable improvement, the MSE of 40.1009 indicates room for further improvement. Additionally, the prediction gain, R_p, of 5.1960dB indicates a considerable reduction in prediction error when compared to the signal variance. Even though this algorithm performs relatively well for a linear model, it is unable to discern non-linearities in the signal. This is why there isn't a perfect alignment with the original signal, as seen in Figure 39 (right).

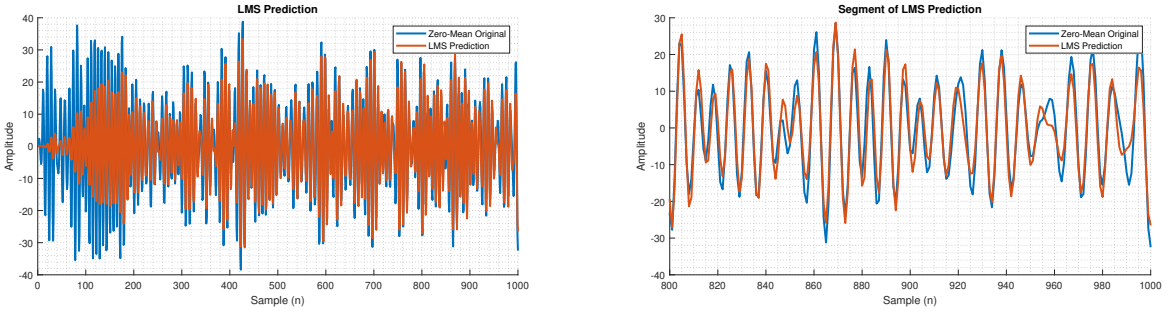


Figure 39: LMS one-step-ahead prediction of zero-mean non-stationary time-series

Typically, the generating process for the data is unknown and non-linear. To address this, a non-linearity, known as an activation function, can be incorporated into the output of the LMS in order for the model to become more expressive. Figure 40 illustrates the application of the *tanh* activation function in the one-step-ahead prediction scenario. Upon interpretation of the results, it becomes evident that the *tanh* function alone is inadequate for this task. With its amplitude range limited to between ± 1, it fails to accurately replicate the signal's wider range, which spans the range ± 40. An MSE of 196.7264 and R_p of -23.1895dB further accentuates this huge disparity between the predicted signal and the true zero-mean signal.

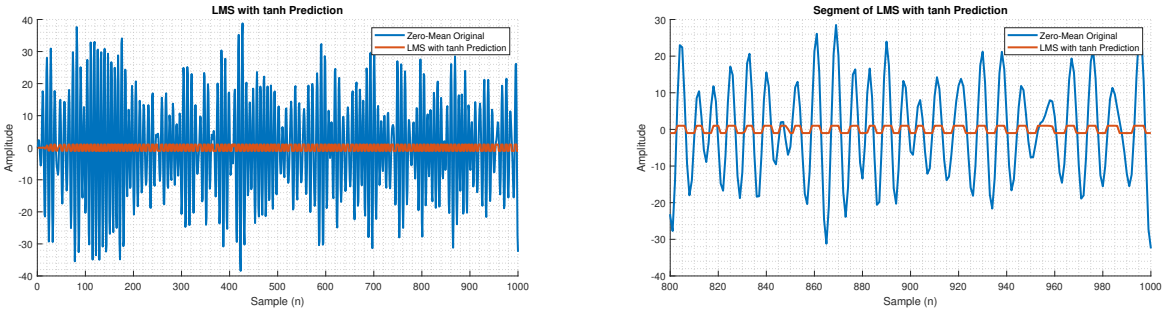


Figure 40: LMS with *tanh* one-step-ahead prediction of zero-mean non-stationary time-series

Figure 40 (right) highlights how the prediction closely follows the trend of the signal but falls short in terms of matching its amplitude, a limitation attributable to the characteristics of the *tanh* function. Hence, utilising the *tanh* function alone proves unsuitable for this task, as its non-linearity fails to adequately capture the signal's amplitude.

To address the limitations of the *tanh* activation function, a scaling factor can be incorporated in the form *a.tanh*. This additional feature allows the *tanh* function to retain its advantageous trend-following properties whilst accommodating the signal's varying magnitudes. For lower values of a (e.g. $a \leq 30$), the amplitude of the prediction gets capped, limiting the perceptron's predictive accuracy, leading to sub-optimal performance. Conversely, for excessively large values, the model's estimation introduces glitches, which again deteriorates the performance of the model. Given that the zero-mean signal varies between ± 40 and $\tanh(x) \in [-1, 1]$, the optimal value of a is found to be $a = 81$, which has an MSE of 4.7909 and an R_p of 16.4897. This model clearly performs better than the standard LMS, boasting an 88% smaller LMS and 217% larger prediction gain. Figure 41 illustrates these results and it's clear to see that the model appears to fit the data well.

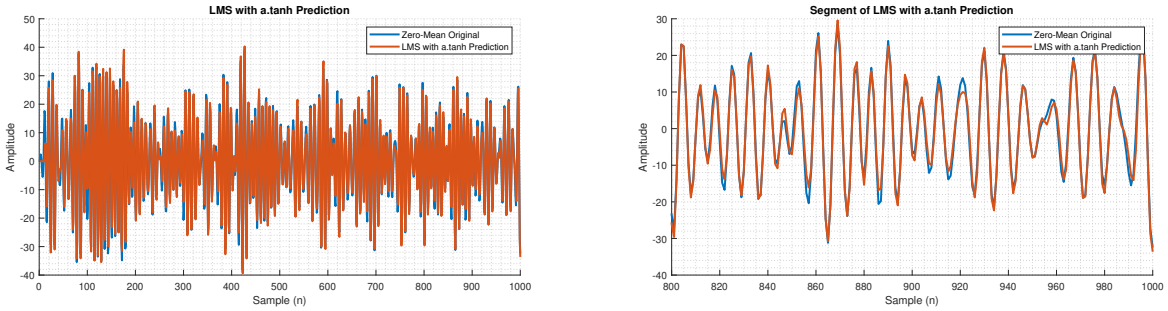


Figure 41: LMS with *a.tanh* one-step-ahead prediction of zero-mean non-stationary time series ($a = 81$)

The incorporation of scaled *tanh* within the LMS framework notably enhances prediction capabilities by effectively capturing non-linearities. However, a key challenge remains in determining the appropriate value for a. This requires prior knowledge of the signal's amplitude range, which may not always be available. This renders this algorithm less suitable for all prediction tasks.

We now address the prediction task concerning the original time-series which exhibits a non-zero mean. The previously discussed methods are tailored for zero-mean data, so in order to accurately predict the original signal, a bias term needs to be introduced, that is $\phi(\mathbf{w}^T\mathbf{x} + b)$, where $\phi(.)$ is the activation function, $\mathbf{w}$ represents the model weights, $\mathbf{x}$ represents the input data and b is the bias. In order to implement this, we augment the input to the algorithm as $[1; \mathbf{x}]^T$, thereby increasing the weight vector by one to accommodate the bias. This addition effectively adapts the algorithm so that $b = w_0 \times 1$, ensuring that the predicted signal acknowledges the non-zero mean of the original signal. Figure 42 illustrates the application of this method, wherein its performance closely resembles that of the previous method without the bias. The primary difference lies in the initial challenge of achieving accurate weights, attributed to the introduction of the bias as an additional consideration.

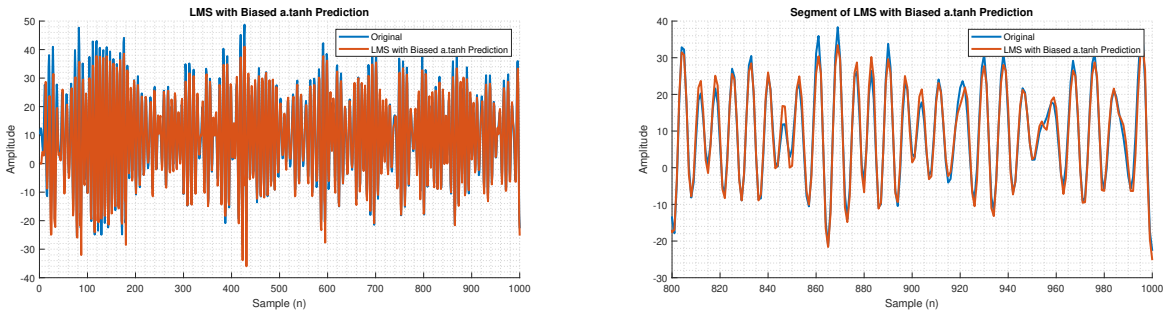


Figure 42: LMS with *a.tanh* one-step-ahead prediction of original non-stationary time-series ($a = 52$)

While the method demonstrates commendable performance in replicating the signal as a whole, as observed in the right plot, it falls short in completely reproducing higher amplitude samples. The MSE of 13.6492 and R_p of 12.2642 indicates a slightly inferior performance compared to its zero-mean counterpart, owing to the added complexity of estimating the bias. Nonetheless, the algorithm achieves a low MSE and high R_p, validating its

effectiveness in the prediction task, albeit with a slower convergence rate compared to its zero-mean counterpart, which is particularly evident in the initials samples depicted in Figure 42 (left).

To address the slow convergence observed in the aforementioned methods, one solution is to pre-train the weights by overfitting to a small number of samples. Specifically, pre-training was carried out with $\mathbf{w}(0)$ set to zero and using 100 epochs to fit the model to the first 20 samples. This pre-training phase yields initial weight values, which are subsequently fed to the LMS algorithm to predict the entire time-series. The results of this approach are depicted in Figure 43.

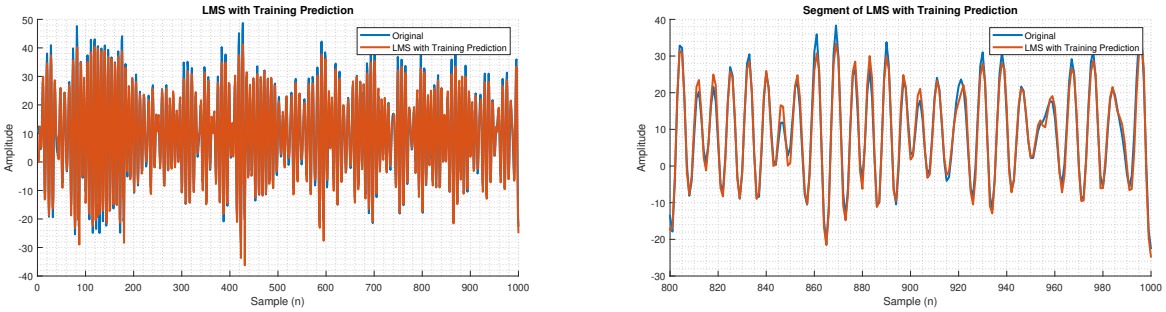


Figure 43: LMS with *a.tanh* one-step-ahead prediction of original non-stationary time-series after pre-training ($a = 52$)

The benefits of this method are evident, in terms of convergence. Unlike previous methods, the initial samples of the signal are accurately estimated, leading to nearly zero convergence time. These benefits have been reaped, whilst maintaining robust prediction capabilities. Overall, the reduced MSE of 6.9398 and increased R_p of 15.1327 validate the efficacy of this improved method over the biased non-linear LMS without pre-training.

In the realm of machine learning, addressing the complexity of real-world problems demands sophisticated models. Traditional activation functions, such as *tanh* often fall short due to their limited expressiveness in capturing non-linear mappings from input to output. To overcome this limitation, researchers have turned to Deep Neural Networks (DNNs) to model intricate relationships in data.

A fundamental algorithm in the training of DNNs is the backpropagation algorithm. This algorithm serves as a cornerstone for optimising the model's parameters to minimise the disparity between predicted and actual outputs. It operates by iteratively propagating error gradients backwards through the network, enabling efficient parameter updates through gradient descent.

The process of backpropagation can be summarised into the following steps:

1. **Forward Pass:** During the forward pass, input data is fed through the network, and activations are computed layer by layer using a chosen activation function like *tanh* or ReLU (Rectified Linear Unit). The output of the network is generated based on the final layer's activations.

2. **Error Computation:** Once the output is obtained, the error between the predicted output and the ground truth labels is computed using a suitable loss function, like MSE for regression tasks or cross-entropy loss for classification tasks.

3. **Backward Pass:** In the backward pass, the error gradients with respect to the parameters of the network are computed using the chain rule of calculus. Starting from the output layer, the gradients are propagated backward through the network, layer by layer.

4. **Parameter Updates:** Finally, the gradients computed during the backward pass are used to update the parameters of the network, typically using an optimisation algorithm such as stochastic gradient descent

(SGD) or one of its variants. The parameters are adjusted in the opposite direction of the gradient to minimise the error.

By iteratively repeating the forward pass, backward pass, and parameter updates over a dataset, the deep network learns to approximate the underlying mapping between inputs and outputs, capturing intricate relationships in the data.

In the context of a DNN composed of multiple dynamical perceptron models arranged in layers, backpropagation enables the efficient training of the network by propagating error gradients through each layer, allowing the network to learn hierarchical representations of the input data. This hierarchical representation enables the model to capture complex, non-linear relationships present in real-world problems, making DNNs suitable for a wide range of applications.

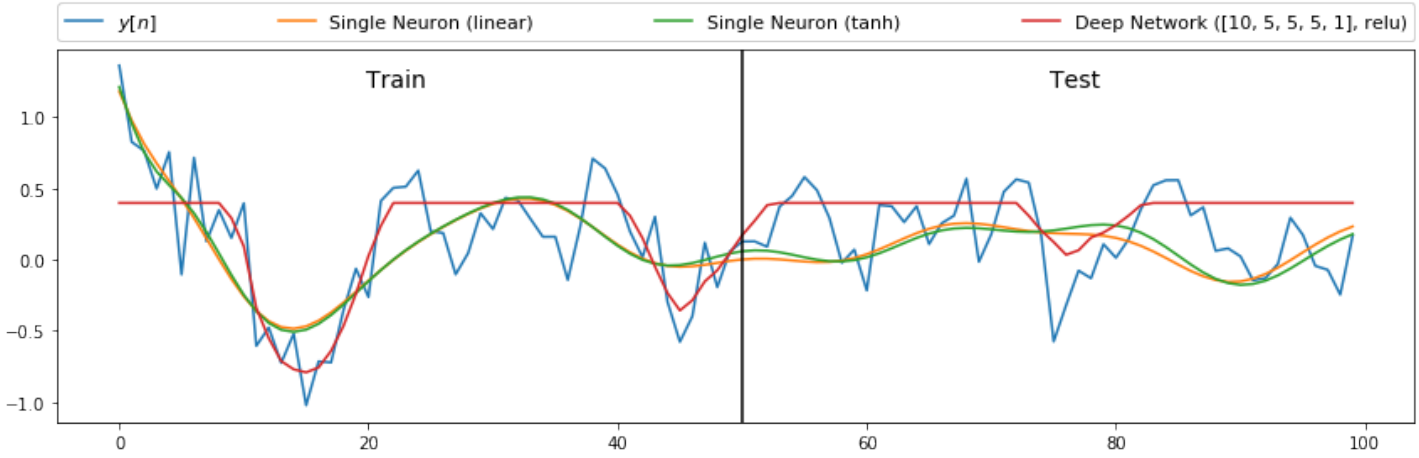


Figure 44: Performance of various predictors on highly non-linear, noise-corrupted time-series ($\sigma_n^2 = 0.05$)

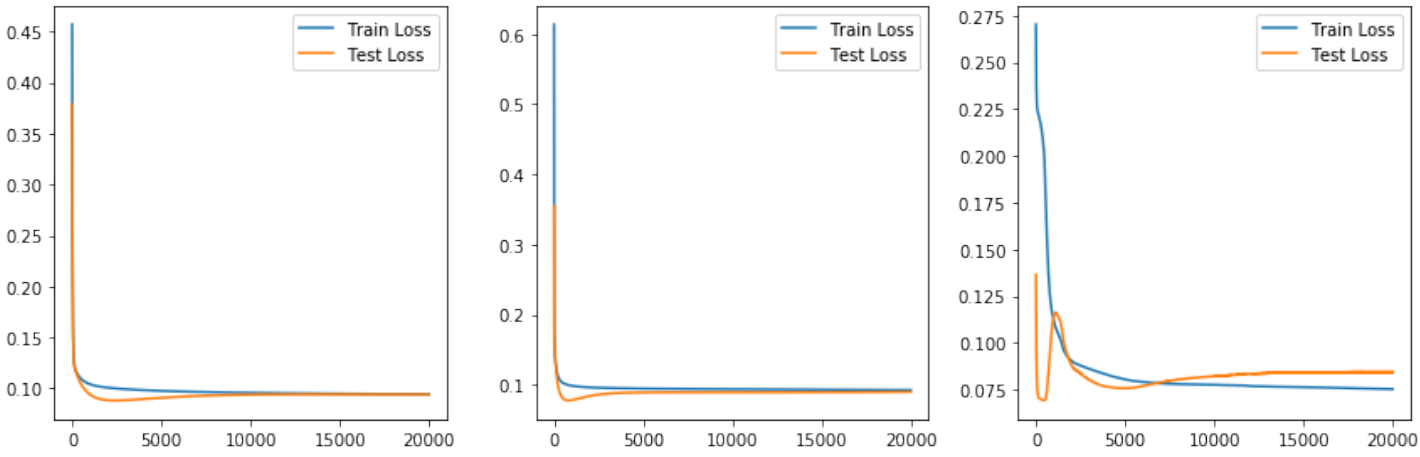


Figure 45: Loss plots for linear single neuron (left), *tanh* single neuron (middle) and DNN (right)

Figure 44 depicts the performance of various predictors on the highly non-linear, noise-corrupted time-series, $y[n]$, with $\sigma_n^2 = 0.05$. The DNN used to train on this time-series was trained on $20,000$ epochs, had 4 hidden layers $(10, 5, 5, 5, 1)$ and used a learning rate of 0.01. Remarkably, as seen in Figure 45, both the linear and *tanh*-activated neurons exhibited comparable performance, attaining a steady-state loss of 0.1 despite the significant non-linearity present in $y[n]$. The primary distinction lies in their convergence rates, with the *tanh* neuron converging faster within $5,000$ epochs, compared to the linear neuron's convergence within $10,000$ epochs.

Conversely, the DNN achieves a lower steady-state loss, registering at 0.075 and ranging from 0.08 to 0.085 for the test set. However, its convergence requires a longer timeframe, spanning $15,000$ epochs, which is expected given its increased complexity. Notably, despite one's propensity to state that there is overfitting, there is none as there is good generalisation during the training phase, disregarding the single random spike in the signal.

Figures 46 and 48 depict the performances of various predictors on the highly non-linear signal, $y[n]$, with noise powers equal 0 and 0.5, respectively. As expected, across all three models, the performance on the noiseless signal surpasses that on the signal with high noise power.

In the low SNR scenario, as illustrated by Figures 46 and 47, the DNN outperforms both of the perceptron models, due to its greater complexity. Whilst both the *tanh*-activated perceptron and linear perceptrons exhibited similar performances, attaining a steady-state test loss of approximately 0.1, the DNN attained a steady-state test loss of 0.03. Notably, the *tanh* model converges much faster within around $7,500$ epochs compared to $17,500$ epochs for the linear model and DNN. Furthermore, for all three models, the steady-state test loss exceeds the training loss. Moreover, despite the single random spike in the test loss for the DNN, none of the models have overfitted.

On the contrary, in the high SNR scenario, as illustrated by Figures 48 and 49, both of the perceptron models outperform the DNN. However, within the first few hundred epochs, both perceptron models reach their minimum and quickly overfit. With regards to the DNN, it overfits around $4,000$ epochs. Even though all three models overfit the noise in the signal, due to the complexity of the DNN, it exhibits the worst generalisation. One should also note that whilst the training losses for the perceptron models achieved a stable steady-state value, the training losses for the DNN exhibited instability, highlighting another issue with applying a complex system to noisy signals.

Overall, the inability of DNNs to generalise makes them a bad choice for the prediction of noisy data because if too much training is done, it leads to overfitting as the model begins to learn the underlying noise as part of the clean signal. However, for the prediction of more complex, noiseless signals, DNNs would be a great choice due to their higher complexity.

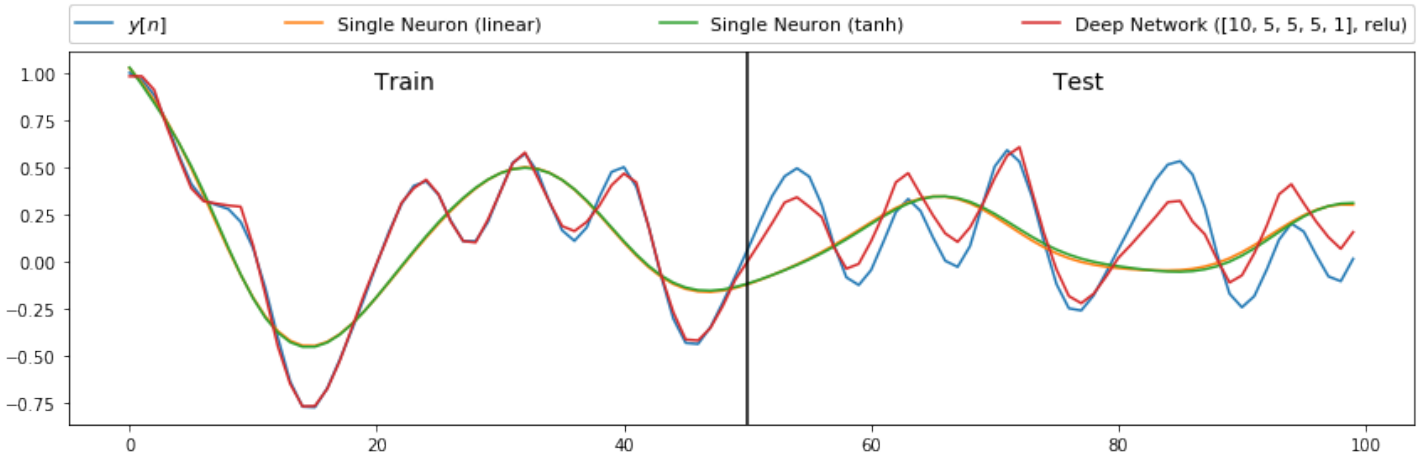


Figure 46: Performance of various predictors on highly non-linear, noiseless time-series ($\sigma_n^2 = 0$)

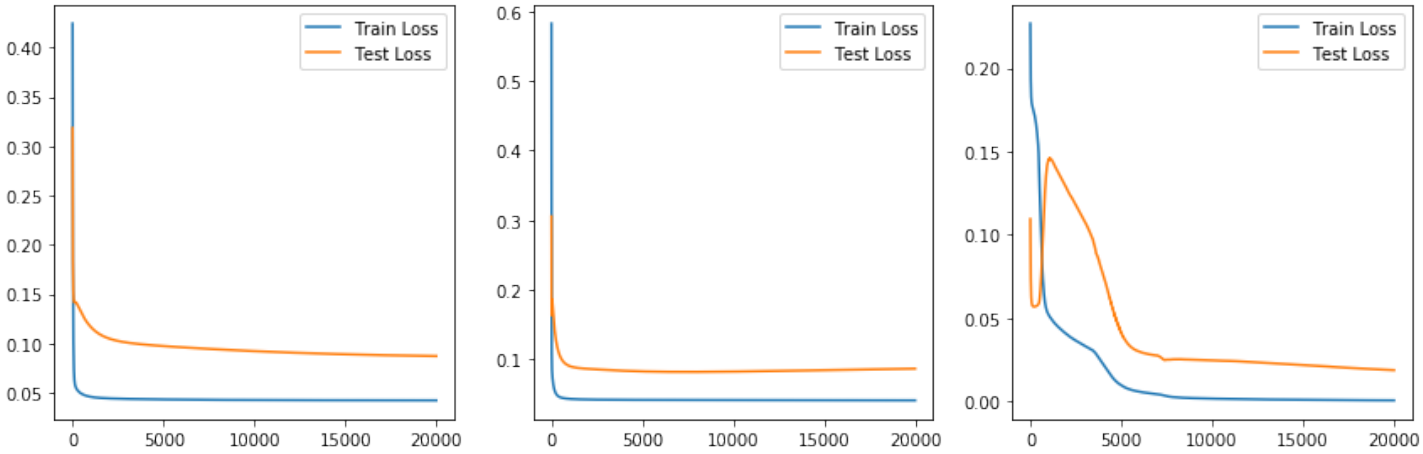


Figure 47: Loss plots for linear single neuron (left), *tanh* single neuron (middle) and DNN (right)

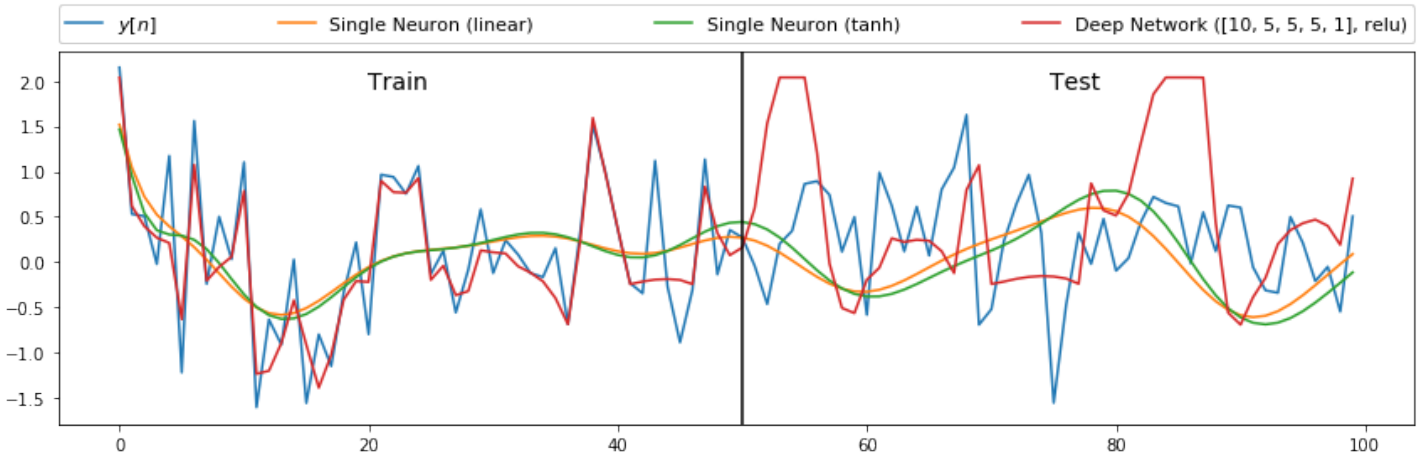


Figure 48: Performance of various predictors on highly non-linear, noise-corrupted time-series ($\sigma_n^2 = 0.50$)

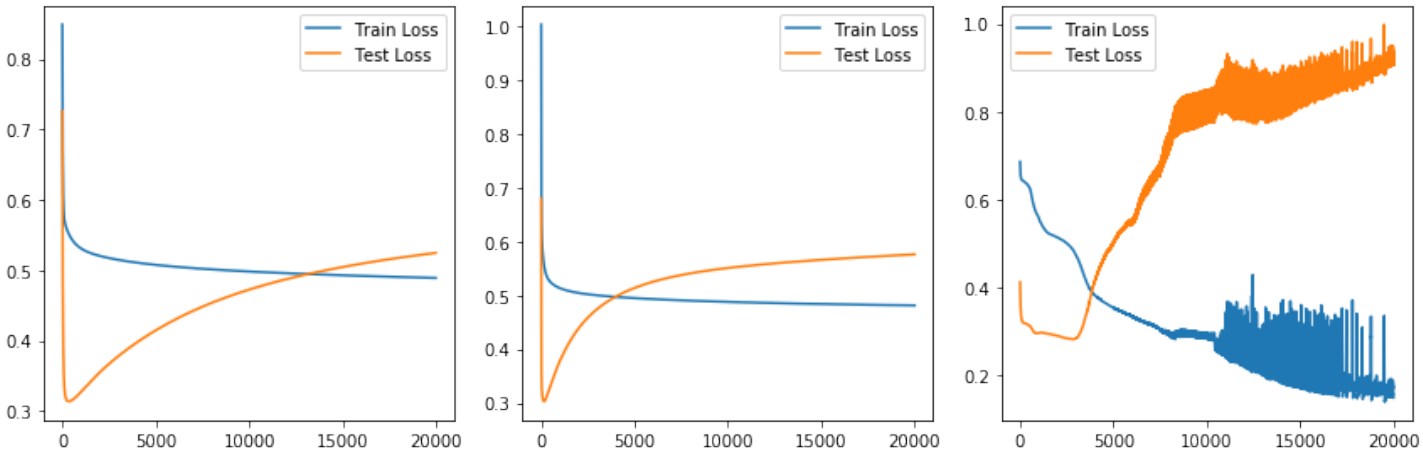


Figure 49: Loss plots for linear single neuron (left), *tanh* single neuron (middle) and DNN (right)

5 Tensor Decompositions and Big Data Applications

This section has been completed in Jupyter Notebooks. Please refer to Part 5 file directory.