Imperial College London

Department of Electrical and Electronic Engineering

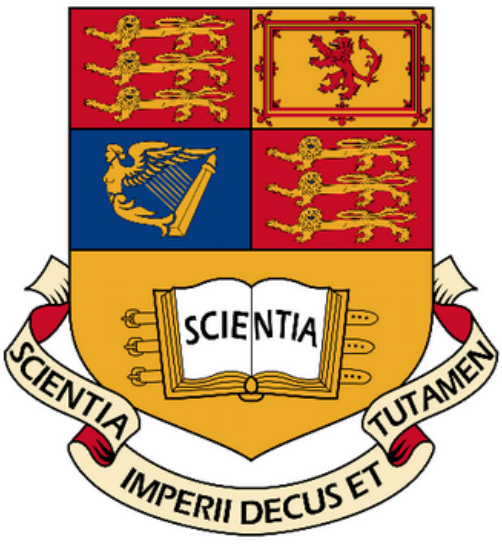
ELEC 97002

Adaptive Signal Processing and Machine Intelligence Coursework

Author:
Charalambos Hadjipanayi

CID:
01077219

Supervisor: Prof. Danilo P.Mandic
April 11, 2020



Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

The equivalence between the two definitions of PSD given in the coursework instructions can be shown analytically by starting from definition 2 (Equation 1.1):

$$P(\omega) \triangleq \lim_{N\to\infty} E\left\{\frac{1}{N}\left[\sum_{n=0}^{N-1} x[n]e^{-jn\omega}\right]^2\right\} \tag{1.1}$$

$$= \lim_{N\to\infty} E\left\{\frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x[n]e^{-jn\omega}x^*[m]e^{jm\omega}\right\} = \lim_{N\to\infty} E\left\{\frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x[n]x^*[m]e^{-j\omega(n-m)}\right\}$$

Since expectation operation and using the assumption that $\sum_n \sum_m E\{x[n]x^*[m]\} < \infty$,

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

By definition, $E\{x[n]x^*[m]\}$ is the autocorrelation function (ACF) of random sequence, which equals the autocovariance function when sequence has zero mean. The ACF can be denoted as $r_x(n,m)$. By assuming that the sequence is wide-sense stationary for the ACF, $r_x(n,m) = r_x(n-m)$, since the ACF will only be a function of the time-lag between the two time instances. Additionally, we can define $k = n - m$ so that:

$$P(\omega) = \lim_{N\to\infty} \frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=n}^{n-(N-1)} r_x(k)e^{-j\omega(k)}$$

By considering the new summation domain due to variable change, and inverting summation order using Fubini's Theorem,

$$P(\omega) = \lim_{N\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{-1}\sum_{n=0}^{k+(N-1)} r_x(k)e^{-j\omega(k)} + \sum_{k=0}^{N-1}\sum_{n=k}^{N-1} r_x(k)e^{-j\omega(k)}$$

$$= \lim_{N\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{-1} r_x(k)e^{-j\omega(k)}\sum_{n=0}^{k+(N-1)} 1 + \sum_{k=0}^{N-1} r_x(k)e^{-j\omega(k)}\sum_{n=k}^{N-1} 1$$

$$= \lim_{N\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{-1} r_x(k)e^{-j\omega(k)}\,[k+N] + \sum_{k=0}^{N-1} r_x(k)e^{-j\omega(k)}\,[N-k] = \lim_{N\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{N-1} r_x(k)e^{-j\omega(k)}[N-|k|] \tag{1.2}$$

$$= \lim_{N\to\infty} \sum_{k=-(N-1)}^{N-1} r_x(k)e^{-j\omega(k)} - \lim_{N\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{N-1} |k| r_x(k)e^{-j\omega(k)} \tag{1.3}$$

Using the mild assumption that ACF decays rapidly, i.e. $\lim_{N\to\infty} \frac{1}{N}\sum_{k=-(N-1)}^{N-1} |k| r_x(k)e^{-j\omega(k)} = 0$, we can obtain definition 1 of PSD (Equation 1.4), where PSD is the Discrete-time Fourier Transform (DTFT) of ACF.

$$P(\omega) = \lim_{N\to\infty} \sum_{k=-(N-1)}^{N-1} r_x(k)e^{-j\omega(k)} = \sum_{k=-\infty}^{\infty} r_x(k)e^{-j\omega(k)} \tag{1.4}$$

The equivalence of the two definitions, known as the Wiener-Khinchin theorem, is thus analytically proven. It is worth noting that PSD is strictly real and positive due the definition 2, which is also satisfied by definition 1 since the Fourier Transform of an even function like the ACF is also real and positive (negative values of ACF are captured by phase spectrum which is zero).

Figure 1.1 shows simulation of the two definitions of PSD for White Gaussian Noise (WGN) with rapidly decaying ACF. The theoretical ACF of WGN is $\sigma^2_{wgn}\delta(k)$, where in this case $\sigma^2_{wgn} = 1$. Any deviations from the two definitions are due to the fact that N is finite (N=100,000) and PSD estimate has high variance. However, the mean value of two definitions is the same, i.e. 0.3331, shown as horizontal lines in plot.

Most real signals lie between WGN and strictly periodic signals. For the case of perfectly periodic signals, ACF is not decaying to zero as N increases, but instead varies periodically, with same frequency as original signal. In order for the two equations to be equivalent, the mild assumption needs to be satisfied by ensuring that $\sum_{k=-\infty}^{+\infty} |k| r_x(k) < \infty$. This can only be achieved in the case that $r_x(k)$ reaches zero value faster than $|k|$ reaches infinity, which is valid for almost all real-life signals and thus the condition is almost always fulfilled in practice. For a perfectly sinusoidal signal, such as $x[n] = sin(n/100)$, however, the theoretical ACF is given by $r_x(k) = (1/2)cos(k/100)$, which causes $\sum_{k=-\infty}^{+\infty} |k| r_x(k) = \infty$. Therefore, $P(\omega) < 0$ and we obtain negative power in the spectrum, causing the two definitions to be non-equivalent. To illustrate that equivalence does not hold for non-decaying ACF's, the two definitions were simulated for $x[n] = sin(n/100)$, producing Figure 1.2. As the plots show, perfectly periodic signals can result in negative power since mild condition is not satisfied, thus for some ω values Equation 1.3 is less than zero. Additionally, the PSD obtained using definition 1 is complex-valued and not purely real. It should be noted that for both WGN and sinusoidal signal, the theoretical ACF equations were used in definition 1. This is because if ACF biased estimates were used, the ACF of sinusoid will decay to zero eventually due to windowing operation and also if unbiased estimates were used, then none will decay to zero.

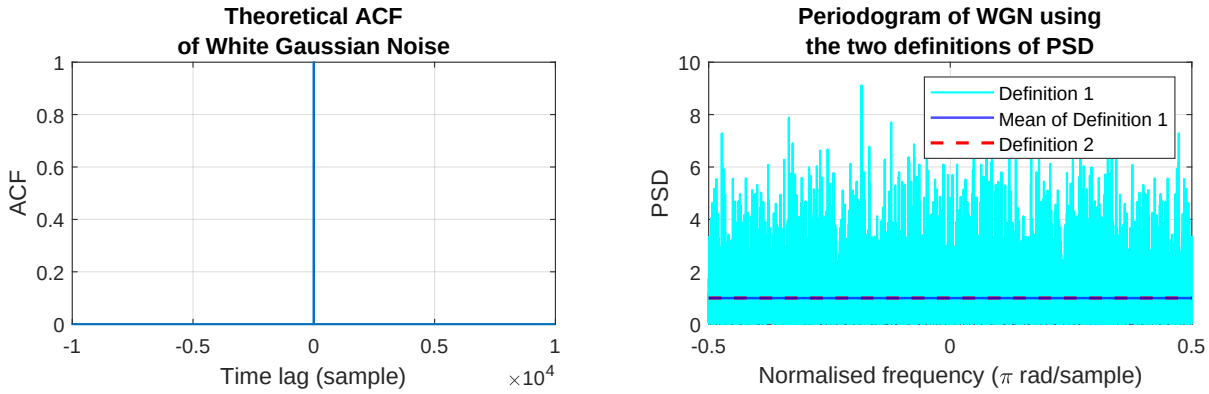


Figure 1.1: Simulation of two definitions of PSD for White Gaussian Noise

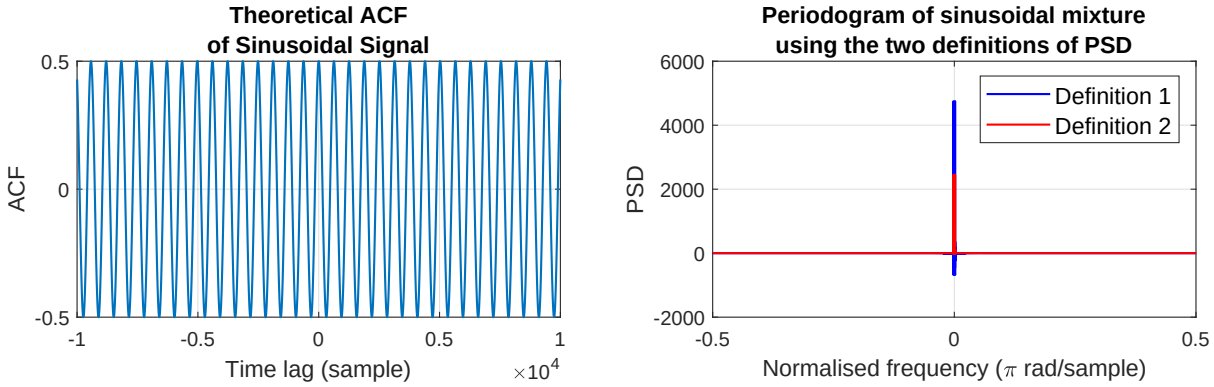


Figure 1.2: Simulation of two definitions of PSD for mixture of sinuoids (N=10,000).

1.2 Periodogram-based Methods Applied to Real-World Data

Part a) - Sunspot Time series

The spectral estimate of sunspot time series was analysed using the modified periodogram technique, where a Hanning window was used in MATLAB function `periodogram()`. Before obtaining the periodogram, a very small DC offset value was added to the data, using `eps` function in MATLAB, to remove any zero-valued points, that will allow application of the logarithm to the data later on. By removing the mean from the data (i.e. the DC component at $\omega = 0$), and linearly detrending the data, allowed low-frequency components of spectrum to be removed so that higher frequency components were more easily identified. The logarithm was also applied to the preprocessed data (with the additional very small offset) and then the mean of logarithmic data was subtracted, in order to identify how perception of the periodicities in the data changes through this process.

As shown in Figure 1.3, centering and detrending data allows peaks in spectrum to be more easily identified. Specifically the sunspot activity is shown to happen maximally at 0.09 Cycles/Year (Period of 11.1 years), due to the presence of a dominant spectral peak at this frequency. Additionally, the first harmonic at around 0.17 Cycles/Year is barely visible in spectrum. By using centered logarithmic data, the spectral peak at 0.09 Cycles/Year is still present and also the first and second harmonics are much more easily identified, at 0.17 and 0.26 Cycles/Year. It should also be mentioned that a peak at 0.01 Cycles/Year is present in the three plots.

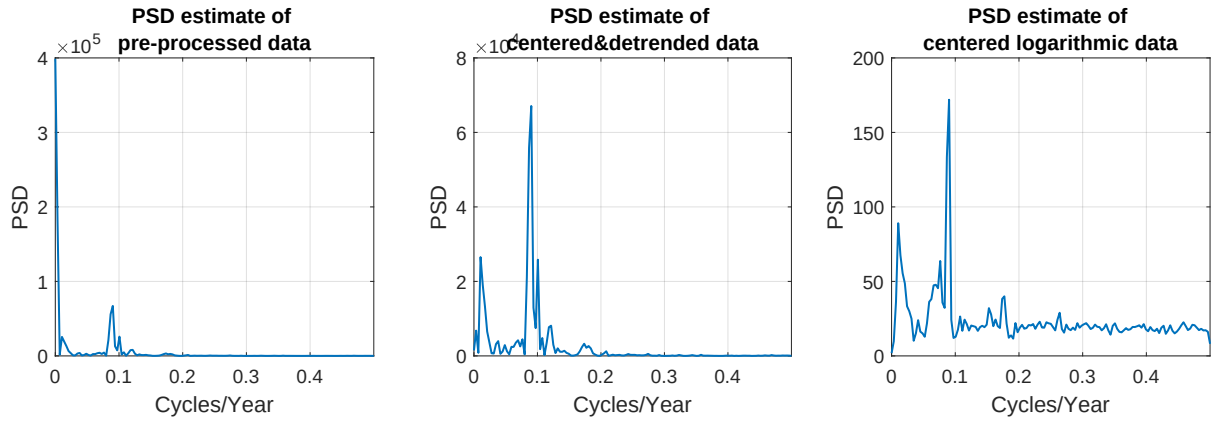


Figure 1.3: (Left) PSD estimate of pre-processed data using `eps`. (Middle) PSD estimate of data when mean and trend are removed. (Right) PSD estimate of logarithmic data with mean value removed.

Part b) - The basis for brain computer interface (BCI)

The spectrum of the electroencephalogram (EEG) signal given was analysed in order to detect the frequency of the flashing visual stimulus observed by the subject which also induced a response in EEG data at the same frequency, known as the steady state visual evoked potential (SSVEP).

Initially the standard periodogram technique (using rectangular window) was applied to the entire recording. Then, Bartlett's method of averaged periodograms was used, with different window lengths ($\Delta t = 10s, 5s, 1s$) . The results are shown in Figure 1.4, where PSD amplitudes were converted to decibel (dB) values for easier peak detection, due to the low amplitude of higher harmonics of the SSVEP fundamental frequency. It is also worth noting that to enable fair comparison across all these spectral analysis approaches, the same number of DFT samples were obtained per Hz. As Figure 1.4, both methods are able to detect the fundamental frequency f_0 (X) at 13 Hz with first and second harmonics at 26 and 36 Hz respectively. There is an additional wide peak at 8-10Hz, which is more clearly detected using Bartlett method, representing alpha-rhythm due to tiredness of the subject. The peak at 50Hz, detected by both methods represents the power-line interference. It should be noted that standard and Bartlett method, specifically using window lengths of 10s and 5s, shows an additional weak peak at 52Hz, representing the third harmonic of the SSVEP.

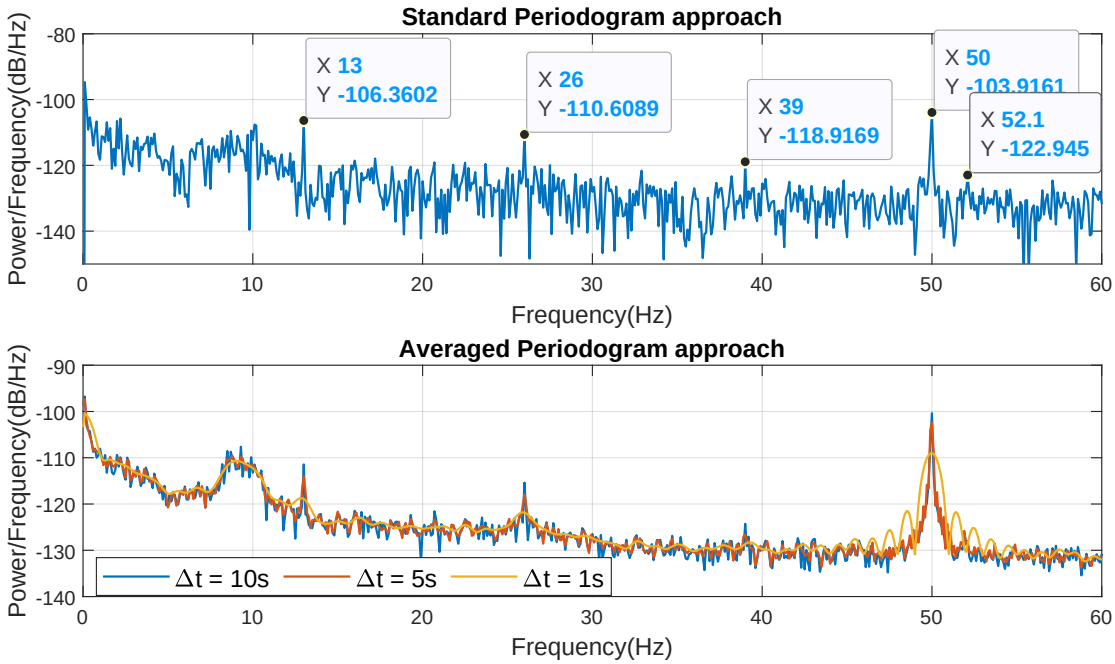


Figure 1.4: (Top) PSD estimate of data using the Standard Periodogram approach. (Bottom) PSD estimate of data using the Averaged Periodogram approach.

The Bartlett method (averaged periodogram) produces clearer peaks in the spectrum for identification, compared to the standard periodogram method. The averaged periodogram of window length of 10s has a lower variance but higher bias (lower spectral resolution) compared to the standard periodogram method.This is because in general the expected value of a periodogram (with rectangular window) is equivalent to the convolution of PSD with the Fourier Transform of the Bartlett window, which has a form of 'aliased' sinc (asinc) squared shape. The main lobe width of this asinc function is inversely proportional to the signal length. Therefore, by decreasing signal length, through truncation via windowing, the mainlobe width increases, masking any nearby spectral peaks thus reducing resolution. Additionally, sidelobe amplitude also increases causing spectral leakage. On the other hand, the variance of periodogram, which is frequency dependent, is proportional to the square magnitude of the PSD at that part specific frequency. However by averaging periodogram M times can reduce variance by a factor of 1/M. As a result the two methods, illustrate the tradeoff between bias and variance of PSD estimates. From Figure 1.5, we can clearly see that standard method has higher variance (factor of 6) compared to a window length of 10s, by higher resolution. However, the peaks can still be identified in the spectrum of the latter method. When window length of 1s is used, variance decreases massively (factor of 60) but the resolution is not sufficient enough to detect second and third harmonics of the SSEVP. Finally, it should be noted that for window length of 1s, a sinc-like shaped peak occurs at 50Hz, which is a residual of the underlying asinc function with very wide main lobe.

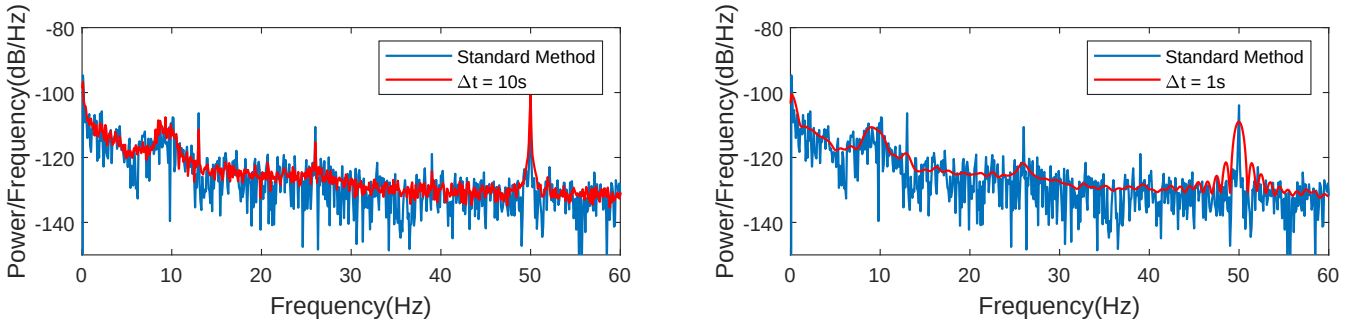


Figure 1.5: Clearer comparison of standard method with window lengths (Δt of 10s and 1s).

1.3 Correlation Estimation

Part a) - Unbiased correlation estimation and preservation of non-negative spectra

Figure 1.6 shows the PSD estimates for WGN, noisy sinusoidal signals and low-pass filtered WGN, obtained using both biased and unbiased ACF estimates, provided by Equations 1.5 and 1.6, for $-N+1 \leq k \leq N-1$.

$$\text{Biased Estimator: } \hat{r}(k) = \frac{1}{N} \sum_{n=0}^{N-|k|-1} x[n]x^*[n+k] \tag{1.5}$$

$$\text{Uniased Estimator: } \hat{r}(k) = \frac{1}{N-|k|} \sum_{n=0}^{N-|k|-1} x[n]x^*[n+k] \tag{1.6}$$

According to Figure 1.6 the ACF estimate of both methods is very similar for values $|k| < 500$. However, for larger values of k, the biased ACF estimate decays to zero, while the unbiased ACF estimate increases in value. Additionally, the PSD estimate based on biased ACF estimator agrees with the expected spectrum shape, which is flat for WGN, impulse train for noisy sinusoidal signal and flat with attenuated high-frequency components for filtered WGN. On the other hand, this is not the case for the PSD estimate from unbiased ACF estimator, since negative values are introduced which would not be present in true PSD.

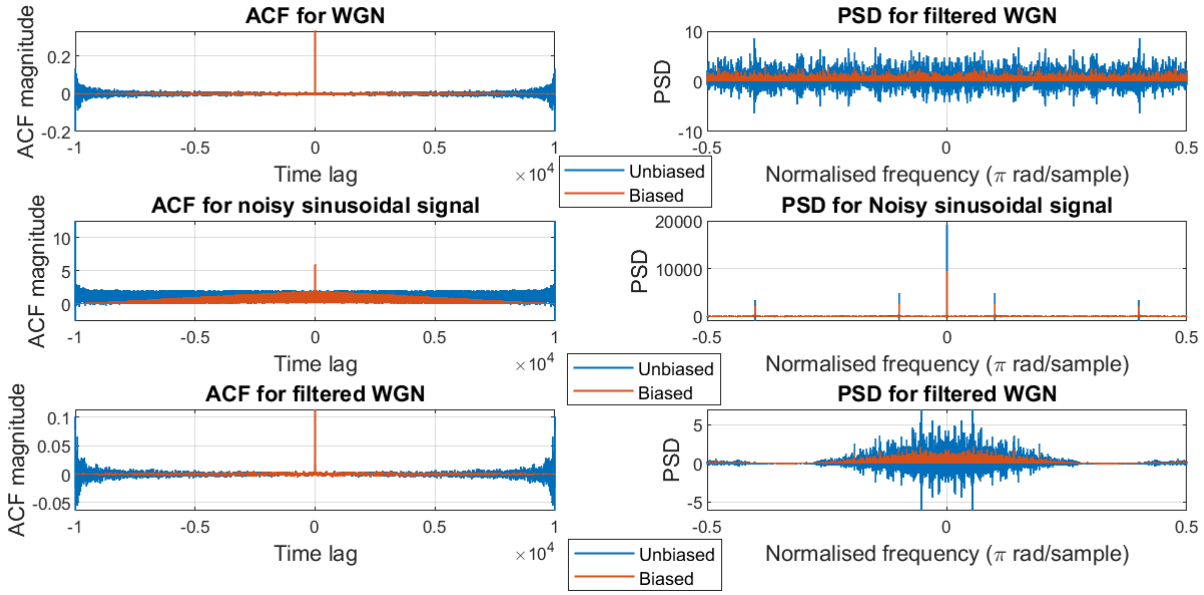


Figure 1.6: Plot of biased and unbiased ACF estimates of WGN, noisy sinusoidal signals and low-pass filtered WGN, with the corresponding correlogram spectral estimates (N=10000).

Although the unbiased ACF estimator seems more appropriate, since its mean matches the true mean of PSD, it can be highly erratic for large lags (k-values) close to N. Due to the factor of $(1/N - |k|)$ involved in the estimator and small number of samples available for large k values, the estimate of ACF may be non-positive semi-definite, meaning that $r(k)$ can be greater than $r(0)$. In other words, the condition that $|r(k)| \leq r(0)$ for $k \neq 0$ may not be satisfied.
The negative values of estimated PSD using unbiased ACF estimator, also shown on Figure 1.6, can explained by considering the expected value of the corresponding periodogram, which is equal to the convolution between the true PSD and the Fourier transform of the rectangular window, given by a sinc-like function $sin(\frac{\omega N}{2})/sin(\frac{\omega N}{2})$. This function can therefore introduce negative values in the PSD estimate. In the case of biased estimate, however, the expected value of the corresponding periodogram is equal to the convolution of the true PSD with the Fourier Transform of the Bartlett window. The Fourier transform of the Bartlett window is given by an asinc function, discussed in Section 1.2b), equal to $sin^2(\frac{\omega N}{2})/sin^2(\frac{\omega}{2})$ which is a strictly positive value.
The presence of negative values can also be thought as a consequence of the equivalence of the two definitions of PSD (Section 1.1) not being satisfied. This is because biased estimator forces ACF to decay to zero for large k values, ensuring that equivalence is satisfied and guaranteeing semi-definiteness of ACF. On the other hand, unbiased estimator may cause ACF to not decay to zero for large k values, thus the estimated PSD can be negative and will not correspond to the true PSD.
For this reason, the biased estimator is widely used, which also has lower variance than unbiased estimator and is also assymptotically unbiased.

Part b) - Multiple realisations of PSD estimate

The PSD estimates of 500 realisations of a random process, obtained using the biased ACF estimator, are shown in Figure 1.7. The random process x[n] consists of a mixture of three sinusoids (0.9, 1.2, 1.5 Hz) corrupted by noise $\eta[n]$, given by Equation (1.7).

$$x[n] = 0.4sin(2\pi n(0.9)) + 0.6sin(2\pi n(1.2)) + 0.5sin(2\pi n(1.5)) + \eta[n] \ , \ where \ \eta[n] \sim \mathcal{N}(0,1) \tag{1.7}$$

The left plot of Figure 1.7 shows that the PSD estimate can on average detect the locations of the three spectral peaks (at 0.9, 1.2 and 1.5Hz). At other locations of spectrum, the mean value converges to a magnitude of 1, representing the power of WGN.The biased ACF estimator is an assymptotically unbiased estimator, which in turn makes the periodogram assymptotically unbiased as well. The expected value of the periodogram is given by convolution of true PSD with the Fourier transform of a Bartlett window, which as N$\rightarrow \infty$ becomes a Dirac delta and thus reduces bias to zero. Therefore, we expect the mean value of the PSD estimate to converge to the true PSD as N increases to infinity. The right graph of Figure 1.7 shows that the standard deviation of the PSD estimate is proportional to PSD value, thus maximum variance occurs at the spectral peaks. In fact, the variance of the periodogram is proportional to the square magnitude of true PSD value, and since it is independent of N, the variance will does not reduce to zero as data length increases. As a result, the periodogram is an inconsistent estimator.

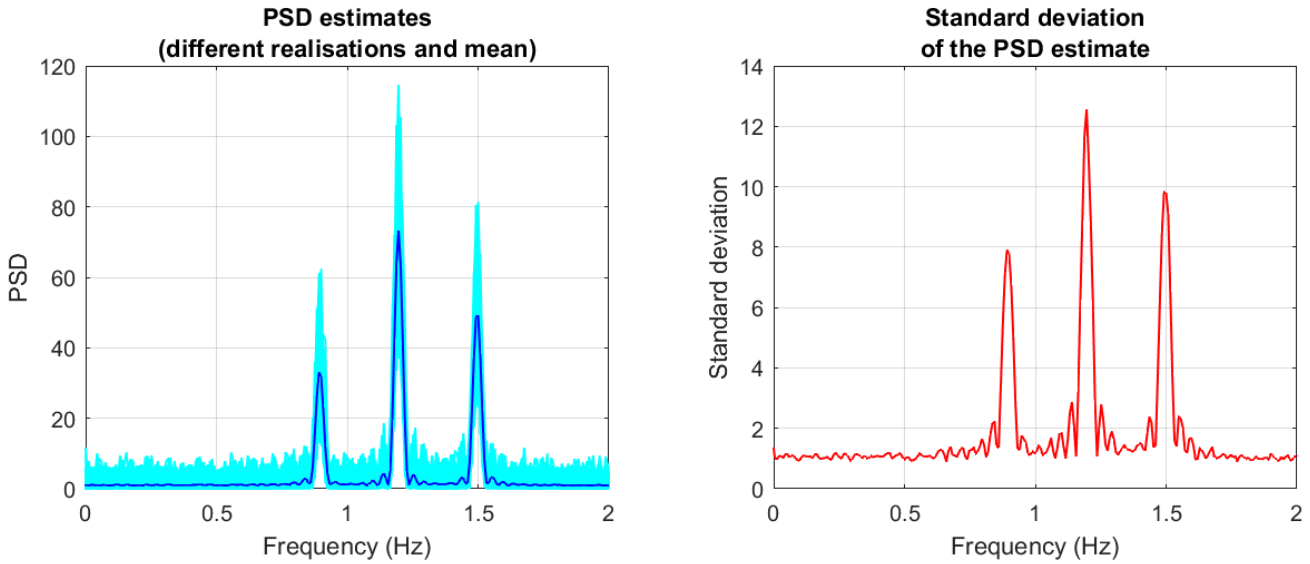


Figure 1.7: PSD estimates of three sinusoids immersed in noise (N=5000). (Left) An overlay of 500 realisations in light blue and their mean in dark blue. (Right) Standard deviation of the 500 estimates.

Part c) - Plotting the PSD in dB

The same estimates as in part b) were plotted in dB (Decibels) together with their associated standard deviation in Figure 1.8.

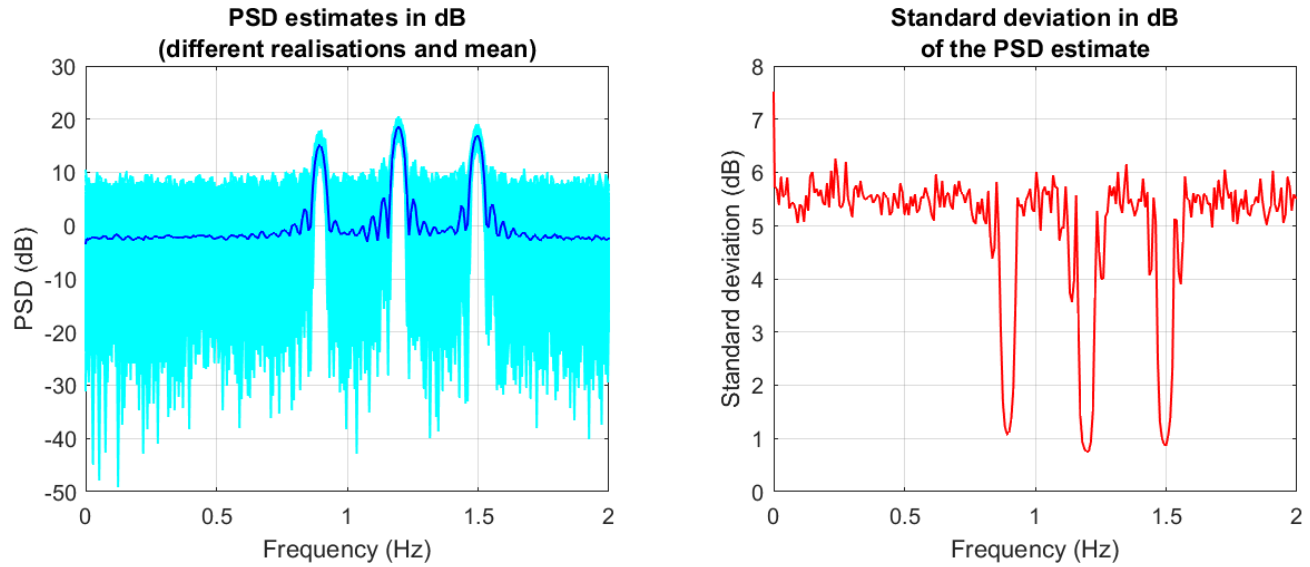


Figure 1.8: PSD estimates in dB of three sinusoids immersed in noise (N=5000). (Left) An overlay of 500 realisations in light blue and their mean in dark blue. (Right) Standard deviation of the 500 estimates.

The $10log_{10}(P(f))$ operation involved in the conversion to dB's, performs a form of smoothing operation on parts of spectrum with largest gradients. This is because if the gradient of original spectrum is $\frac{dP(f)}{df}$, then the gradient of the log-spectrum is equal to $\frac{dP(f)}{df}\frac{1}{P(f)ln(10)}$, which means that log-spectrum gradient is inversily proportional to the PSD value at that frequency. As a result, the gradient at spectral peaks gets attenuated (smoothed out), while fluctuations due to noise are amplified. This method allows easier detection of spectral peaks since the variation between different realisations is reduced due to smoothing operation. This is also identified in Figure 1.8, where the standard deviation is minimum at these peaks, compared to Figure 1.7. Another benefit, is that this method can allow clearer detection of other low-amplitude signals in spectrum which may not be visible in the normal spectrum, especially in the case when one signal component has significantly larger amplitude.

Part d) - Generation of complex exponential signals

A complex-valued signal was generated, composed of two complex exponential signals with frequencies of 0.3Hz and 0.32Hz corrupted by complex WGN with zero mean and variance of 0.2, expressed in equation 1.8.

$$x(n) = e^{j2\pi(0.3)n} + e^{j2\pi(0.32)n} + \eta(n)\ ,\ where\ \eta(n) \sim \mathcal{N}(0, 0.2) \tag{1.8}$$

The spectral estimate of the signal for different signal lengths (N) is shown in Figure 1.9. This shows that for signal lengths of N=20 and N=30, the periodogram was not able to identify the two lines in the spectrum corresponding to the two frequencies, due to periodogram resolution (which is proportional to 1/N) being greater than the separation of frequencies. By considering more data samples (> 30), the periodogram begins to show the correct line spectra. More specifically, the periodogram is biased by Bartlett window, since its expected value is a convolution between the true PSD and Fourier Transform of Bartlett window, which is a sinc-like positive function (asinc), expressed as $W_B(\omega) = sin^2((\omega N)/2)/sin^2(\omega/2)$. The width of the main lobe of this sinc-like function introduces spectral estimate smoothing, resulting in masking of closely spaced narrow-band signal components. The main lobe width is equal to $2f_{sampling}/N$, therefore to allow separation of peaks by the width of window's main lobe, $N \geq 100$, since frequency separation is 0.02Hz. However, in practise the effective mainlobe width is roughly 0.89/N for rectangular window, corresponding to the main lobe width with amplitude greater than -13dB, compared to main lobe peak value (0dB). This is because the amplitude of first side lobe of window has amplitude of -13dB. This is usually referred to as the -13dB width. In this case, $N \geq 45$, to allow separation of the frequencies with window amplitude difference greater than 13dB. For lower lengths than N=45, alternative methods should be considered, e.g. subspace method.

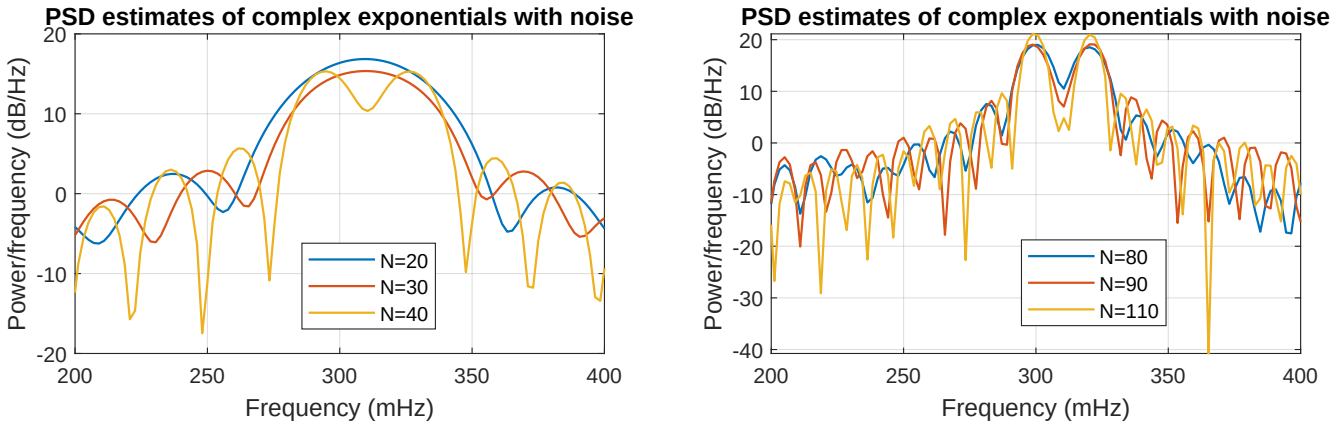


Figure 1.9: Spectral estimates of complex-valued signal described by equation 1.8, using the periodogram method with rectangular window, unit sampling rate and 512 frequency bins.

Part e) - Frequency estimation by MUSIC

The code provided below, allows implementation of the Multiple Signal Classification (MUSIC) algorithm that allows estimation of signal spectrum and is refered to as a type of Subspace method. This method performs an eigenanalysis on the autocorrelation matrix (ACM) of a time series and assumes that the underlying signal consists of a linear combination of p complex exponentials in WGN.

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

The first line of code (`corrmtx()`) obtains an estimate of the autocorrelation matrix (`R`) for the input signal `x`, where `x` is a linear combination of complex exponentials corrupted by WGN. For signal length N, the number of lags `m` was set to a value of `(N/2)-1=14` to ignore errors in autocorrelation calculation at higher lag values. The Toeplitz data matrix `X` (2(N-m)-by-(m+1)) returned by the first line, is constructed based on the modified covariance method (forward-backward method), so that autocorrelation matrix `R=(X'*X + (X'*X)')/2` is an unbiased estimate of the true ACM, with more accurate estimates and no windowing being applied to data (as compared to the default method `'autocorrelation'`).

The second line of code implements the function `pmusic()`, which computes MUSIC algorithm and returns `S` which is the pseudospectrum estimate of the input signal `x`. Under the assumption of uncorrelated complex exponentials and WGN, the ACM of signal $\mathbf{R}_{xx}$ can be expressed as a sum of ACM of noiseless complex exponential signal $\mathbf{R}_s$ and ACM of noise $\mathbf{R}_n$, i.e. $\mathbf{R}_{xx} = \mathbf{R}_s + \mathbf{R}_n = \mathbf{EPE}^H + \sigma_w^2\mathbf{I}$, where $\mathbf{E}$ is an Mxp matrix (M=signal length) containing the p signal vectors $\mathbf{e}_1$ to $\mathbf{e}_p$ with $\mathbf{e}_i = [1, e^{j\omega_i}, ..., e^{j\omega_i(M-1}]^T$, $\mathbf{P} = diag\{P_1, .., P_p\}$ is a diagonal matrix containing the signal powers P_i, σ_w^2 is the variance of WGN and $\mathbf{I}$ is the identity matrix. Since the rank of $\mathbf{R}_s$ is equal to p, it means that $\mathbf{R}_s$ only has p non-zero eigenvalues. As a result, the ACM of signal $\mathbf{R}_{xx}$, has eigenvalues $\lambda_i = \lambda_i^s + \sigma_n^2$, where λ_i^s are the p non-zero eigenvalues of $\mathbf{R}_s$. Therefore, MUSIC algorithm first performs an eigen-decomposition of ACM of signal, arranging the eigenvalues in order of decreasing order. Since the rank of $\mathbf{R}_s$ is p, the eigenvalues of ACM are separated into those being greater than σ_n^2, which span the p-dimensional signal subspace, and those equal to σ_n^2, spanning the (M-2)-dimensional noise subspace. Since ACM of signal is Hermitian, the signal subspace and noise subspace are orthogonal. The signal vectors $\mathbf{e}_i$, for i=1,..p, lie within the signal subspace , which is spanned by the signal eigenvectors, which means that inner product of signal vectors and noise eigenvectors ($\mathbf{v}_k$ for k=p+1,...,M)

is zero. Using this property the MUSIC frequency estimation function is given by:

$$\hat{P}_{MU}(\omega) = \frac{1}{\sum_{k=p+1}^{M} |\mathbf{e}^H \mathbf{v}_k|^2} \tag{1.9}$$

where the PSD estimate, referred to as Pseudospectrum, will have p spectral peaks corresponding to the p complex exponentials within the signal. Concerning inputs to function `pmusic()`: `R` is the input ACM estimate, `2` is the dimensionality of signal subspace, `[]` uses the default integer length of the FFT which is 256, `1` specifies the sampling frequency of signal, where we use a value of 1 to obtained a normalised frequency axis (`f`), `'corr'` specifies that input to function is an ACM estimate. Since the signal vector $\mathbf{e}_i$ consists of complex exponentials, it means that its inner product with noise eigenvectors amounts to a Fourier transform, which is why MATLAB uses FFT for this method. The third line, of the code plots the Pseudospectrum against the normalised frequency axis and displays plots for normalised frequency range of 0.25 and 0.40.

The PSD estimate using MUSIC method for the process described by Equation 1.8, using data length of N=30, is shown in Figure 1.10, which allows comparison with the standard periodogram method shown in Figure 1.9. This shows that while periodogram is unable to separate the two spectral peaks due to to spectral masking (high bias), the MUSIC algorithm is successful in identifying the two closely-located peaks at 0.30 and 0.32, for low data lengths (form of superresolution). MUSIC method has genereally lower bias (higher resolution) than periodogram estimate, which decreases as SNR increases and also exploits the underlying data structure.

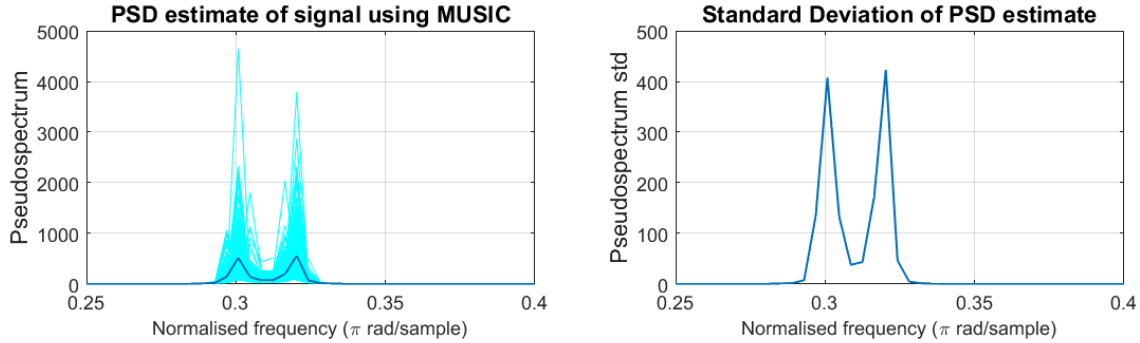


Figure 1.10: (Left) Spectral estimates of complex-valued signal described by equation 1.8, using the MUSIC method for N=30, using 500 realisations of the signal. The mean is shown with a thick blue line. (Right) Standard deviation of MUSIC method.

One of the main disadvantages of MUSIC method is that it requires that parameter p is known in advance or estimated on the basis of eigenvalue distributions. Also, as Figure 1.10, MUSIC PSD estimate has significantly higher variance than the periodogram, although variance of both is still proportional to the PSD value at that frequency. Furthermore, the magnitude of the Pseudospectrum only shows on average the relative magnitudes of the two exponentials (less accurate in providing relative magnitude of components). However, as SNR increases, bias on peak locations decreases, but variance increases and this affects the relative magnitudes of the spectral peaks. This means you can find peak locations more accurately, but the relative proportions of signal components become less accurate. What is more, for low SNR values MUSIC can generate false peaks in spectrum. The computational complexity of MUSIC algorithm is considerably higher than that of periodogram and if parameter p requires to be estimated using MDL or AIC criteria, complexity increases further.

1.4 Spectrum of Autoregressive Processes

Part a)

The AR parameters $\mathbf{a} \in \mathbb{R}^p$ of an Autoregressive (AR) Process of order p, can be obtained from the Yule-Walker equations. The relation between the ACF and model parameters is given by equation 1.10.

$$\mathbf{R}_{xx}\mathbf{a} = \mathbf{r}_x \quad \Rightarrow \quad \mathbf{a} = \mathbf{R}_{xx}^{-1}\mathbf{r}_x \tag{1.10}$$

where $\mathbf{R}_{xx}$ is the autocorrelation matrix of signal of interest x[n]. For AR parameters to be determined, the autocorrelation matrix must be invertible. The ACF matrix Rxx, when biased estimator is used is positive definite, symmetric Toeplitz matrix, which guarantees matrix inversion, allowing AR parameters to be determined. This is because biased estimator ensures that $r_x(0) > r_x(k)\ \forall k \neq 0$, thus guaranteeing that ACF is positive definite. As a result, all the eigenvalues of the ACF matrix will be positive, which means that $\lambda = 0$ is not an eigenvalue of $\mathbf{R}_{xx}$ and thus inverse always exist. On the other hand, when unbiased ACF estimate is used, the ACF will not decay to zero for large lag values and as a result the condition for positive-definiteness may not be satisfied, as we saw in Figure 1.6. This in turn, may cause presence of non-positive eigenvalues of ACF matrix, which means that $\mathbf{R}_{xx}$ can potentially be non-singular, so we will not be able to obtain AR coefficients in this case.

Part b)

An AR(4) process, given by equation 1.11, was generated in MATLAB using 1000 samples (N=1000), from which the first 500 were discarded to remove transient output of filter.

$$x[n] = 2.76x[n-1] - 3.81x[n-2] + 2.65x[n-3] - 0.92x[n-4] + w[n] \quad , \; wherew[n] \sim \mathcal{N}(0,1) \tag{1.11}$$

The PSD estimate of signal was calculated using the function `pyulear()` in Matlab which obtains the power spectral density estimate, using the Yule-Walker method. The PSD estimate was obtained for different model orders p=2,...,14 and were compared to the True PSD value. Results are shown in Figure 1.11.

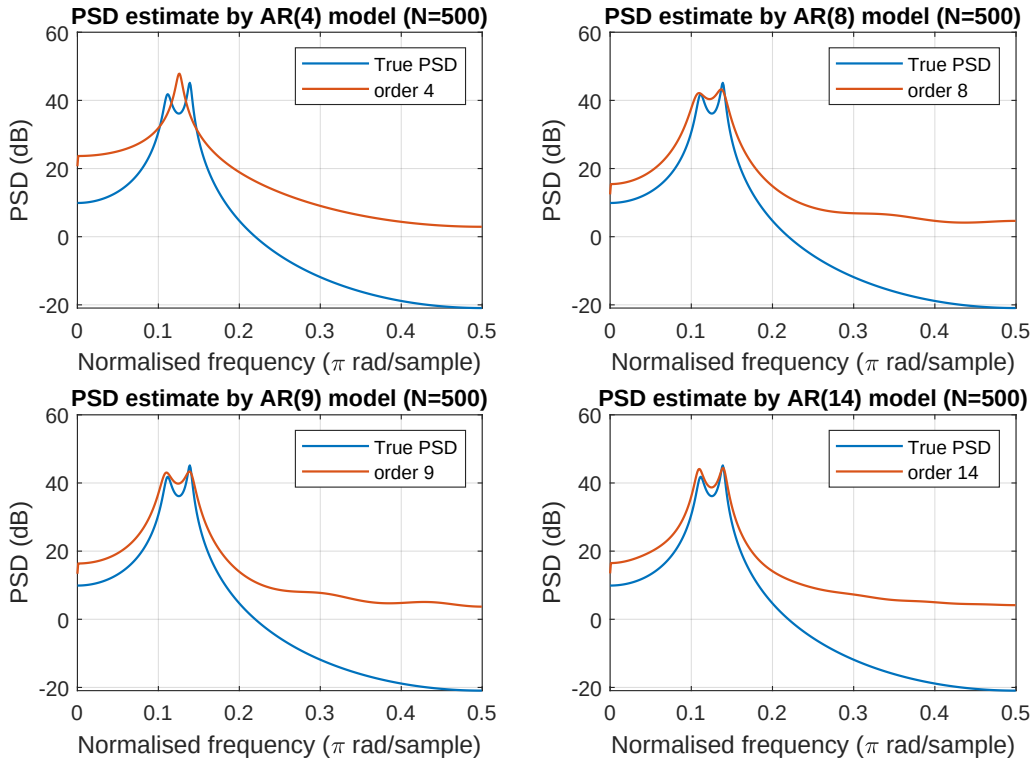


Figure 1.11: Spectral estimates by AR models of different orders for signal of length 500.

Figure 1.11 shows that low AR model order, such as AR(4) which matches the underlying model, the estimates are not successful in detecting the two peaks that are present in the true PSD, but only a single peak. As model order increases, however, the spectral estimates provide better estimates of the true PSD and capture both peaks in spectrum. In general, the greater the order of AR model (over modelling), the greater its accuracy (prediction error decreases) since the model has sufficient degrees of freedom to adequately capture the full underlying structure of the data. This occurs until a plateau is reached in the error, as shown in Figure 1.12, where in fact the minimum mean squared error (prediction error) occurs at model order p=9. On the other hand, as model order increases the higher its complexity. The tradeoff between computational complexity and model accuracy can be established by introducing a "penalty" for higher order models. Specifically by using the Akaike information criterion (AIC) or the corrected AIC(AIC_c), which is better suited for small sample sizes, the optimal orders are p=6 and p=2 respectively.
The fact that AR model performs poorly at p=4, can be explained due to the bias of the ACF which is high due to the small sample size. Performance of model will be enhanced if a greater sample is used (Part c) or greater model order ($p > 4$) is used.

Part c)
The same experiment is repeated as in part b) for data length of 10,000 samples. The comparison between the PSD estimate obtained for different model orders p=2,...,14 and True PSD value are shown in Figure 1.13. Figure 1.12 shows how the MSE of prediction changes as a function of model order, for data lengths of N=1000 and N=10,000, with red circles representing model orders with minimum error.

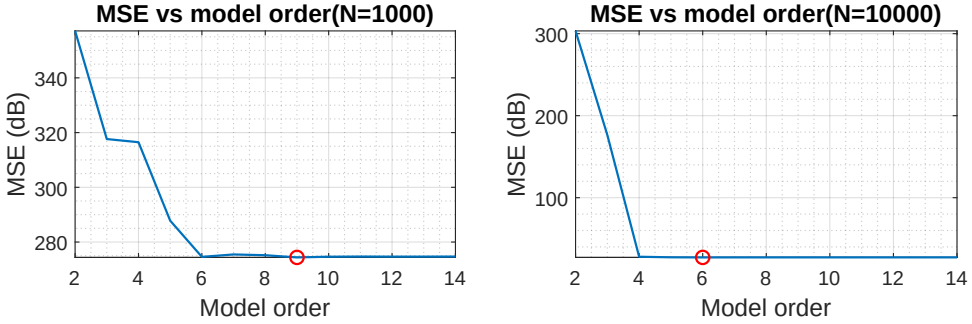


Figure 1.12: MSE as a function of AR model order, (Left) for N=1000 and (Right) for N=10000.

For this case, the accuracy of estimates has significantly improved since an AR(4) model could successfully detect both peaks in spectrum. This can be explained by the reduction of bias in the estimation of AR parameters. Additionally, as Figure 1.12 the MSE has been reduced for all model orders and its minimum value if found at order 6. Furthermore, by considering the tradeoff between computational complexity and model accuracy, as was done in part b), AIC and AIC_c have both optimal orders of p=4. This agrees with the correct AR(4) model order.

Model orders ($p < 4$) lower than the correct AR(4) model order (under-modelling) do not have sufficient degrees of freedom to adequately capture the full underlying structure of the data, resulting in high MSE errors. On the other hand, as model order increases above true AR(4) model order (overfitting), the MSE plateaus since the two spectral peaks are very closely captured. However, as model order $p > 6$, the complexity increases and a slight increase in MSE is also detected.

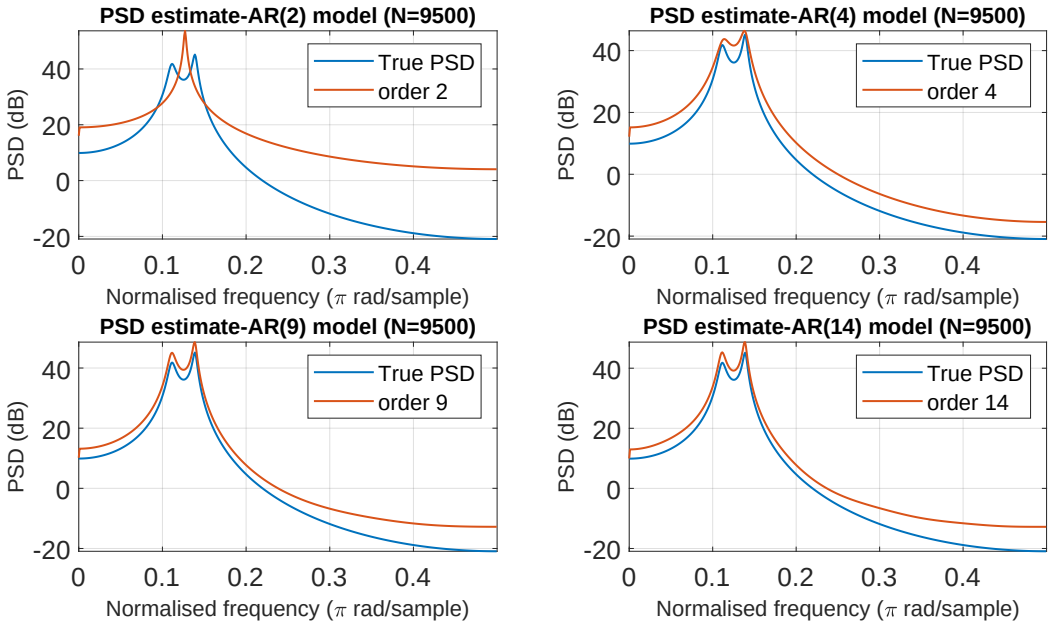


Figure 1.13: Spectral estimates by AR models of different orders for signal of length 9500.

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

Part a)

Respiratory sinus arrhythmia (RSA) is the modulation of cardiac function synchronised with respiration, where the R-R intervals (RRI) on an ECG recording is shortened during inspiration and prolonged during expiration. Therefore, in order to study this phenomenon, the RRI's of the recorded ECG where obtained, from three trials: for normal breathing rate, breathing rate of 25bpm and breathing rate of 7.5bpm. The PSD estimates of the RRI data were then obtained using the standard periodogram and averaged periodogram using non-overlapping rectangular windows of lengths of 50s and 150s, as illustrated in Figures 1.14-1.16.

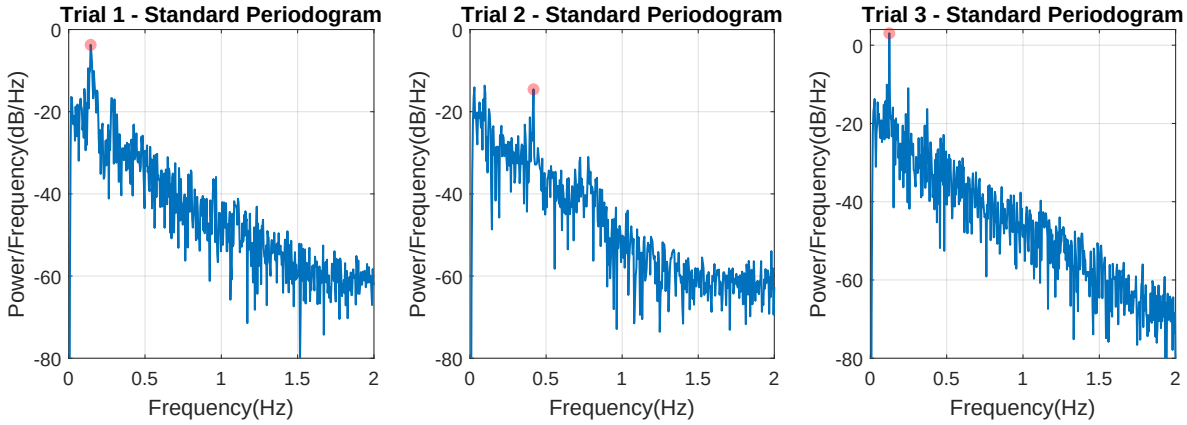


Figure 1.14: PSD estimate of the three trials using standard periodogram, with highlighted peak locations.

Part b)

By analysing the spectrum of RRI data using the standard periodogram, the peaks corresponding to respiration rates of the three experiments were identified. As Figure 1.14 shows, trial 1 spectrum has a peak at 0.144Hz (8.64 breaths/minute), trial 2 spectrum a peak at 0.417Hz (25.02 breaths/minute) and trial 3 spectrum a peak at 0.124Hz (7.44 breaths/minute). Additionally, the first harmonic can also be identified for trial 1 and trial 2, while the first three harmonics can be seen in spectrum of trial 3.

As Figures and show, the averaged periodogram method reduces variance of PSD estimates, while making the spectral peaks broader due to higher bias (lower resolution due to spectral leakage). Specifically, there is a bias-variance tradeoff, where variance decreases and bias increases when window size decreases. Also, as window size decreases the amplitudes of each spectral peak decrease.

As window size decreases from 100s to 50s, the variance of PSD estimate drops, allowing spectral components with lower amplitudes to be identified more easily, which are not easily distinguished in larger window lengths. However,

the exact location of these spectral components cannot be reliably obtained, due to high bias. For example, for trial 1 the peak in standard periodogram appears at 0.144Hz (8.6bpm) while for 50s window appears at 0.148Hz (8.9 bpm).

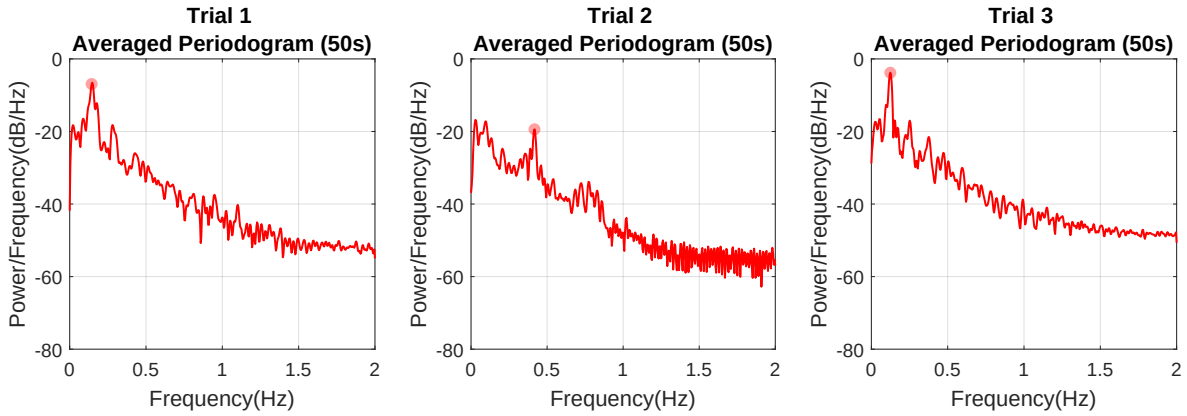


Figure 1.15: PSD estimate of the three trials using average periodogram with highlighted peak locations, using window length of 50s.

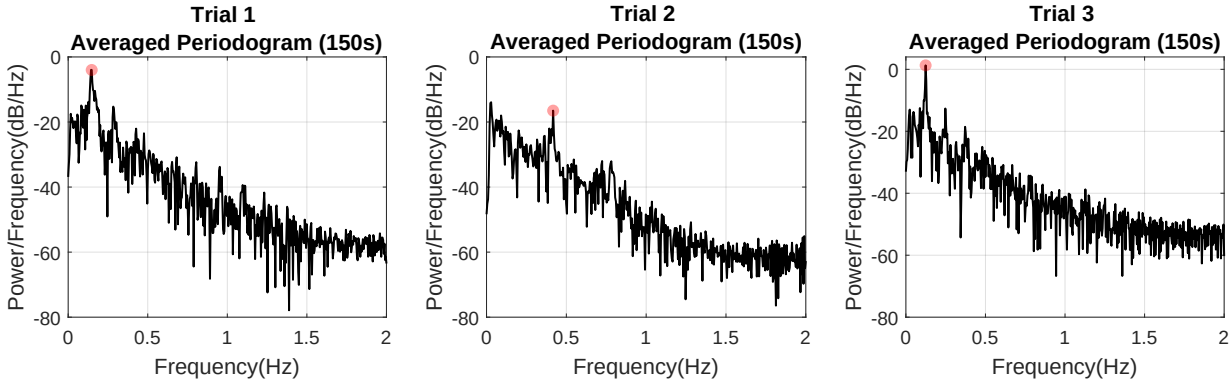


Figure 1.16: PSD estimate of the three trials using average periodogram with highlighted peak locations, using window length of 150s.

Part c)

The AR spectrum estimate of the RRI signals for the three trials is shown in Figure 1.17, where the optimal AR model order was obtained by increasing order until a peak in spectrum was observed, corresponding to the theoretical respiration rate.

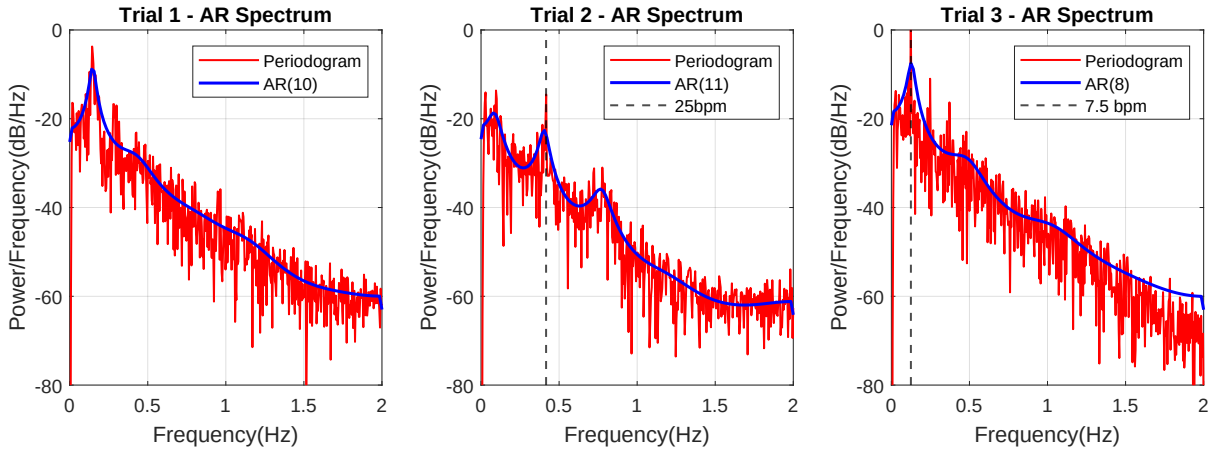


Figure 1.17: Comparison between AR and standard periodogram spectral estimates for the three trials. Trial 1 modelled by AR(10), Trial 2 by AR(11) and Trial 3 by AR(8). Dashed lines in second and third plot correspond to expected locations of peaks (at 25bpm and 7.5 bpm)

As Figure 1.17 shows, AR method provides a smooth and clearer spectrum than periodogram method, that allows easier peak identification due to the sharp peak located at frequency corresponding to respiration rate. The AR spectrum does not suffer from bias-variance trade-off or windowing, however it depends significantly on choice of model order. If the chosen order is too high (over-fitting) AR method can give rise to false or spurious peaks in addition to increased computational complexity, due to additional poles in the model. On the other hand, if model order is too low, then spectrum will be highly smoothed, with very low resolution that lacks detail and spectral peaks may not be visible. Furthermore, in order to identify the harmonics in spectrum, the model order must increase significantly above the chosen optimal model order. This is because AR method ignores harmonics and noise components and only highlights the main peaks. For shorter data segments, e.g. when compared to 50s window averaged periodogram, AR

spectrum has better spectral resolution, since the bias-variance trade-off of periodogram-based methods may not be easily balanced in a way to produce acceptable resolution to the spectrum.

1.6 Robust Regression

Part a)

Singular Value Decomposition (SVD) was obtained for matrices $\mathbf{X}$ and $\mathbf{X}_{noise}$ and the singular values of these matrices are shown in Figure 1.18. The square error between each singular value of $\mathbf{X}$ and $\mathbf{X}_{noise}$ is also shown in Figure 1.19.

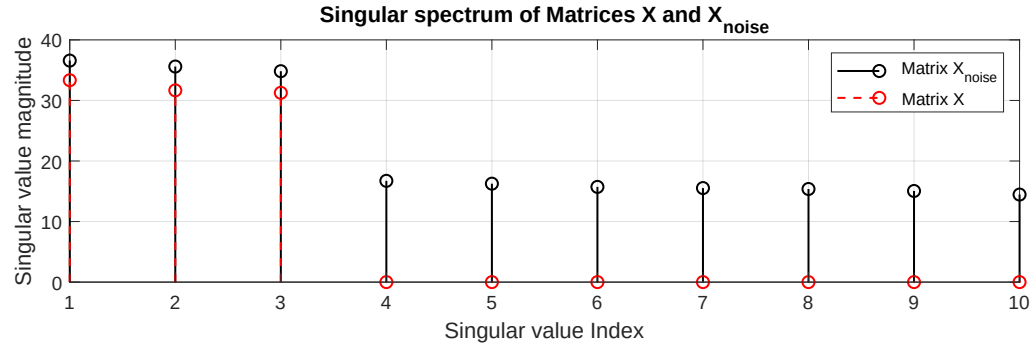


Figure 1.18: Singular Spectrum showing the singular values of matrices $\mathbf{X}$ and $\mathbf{X}_{noise}$

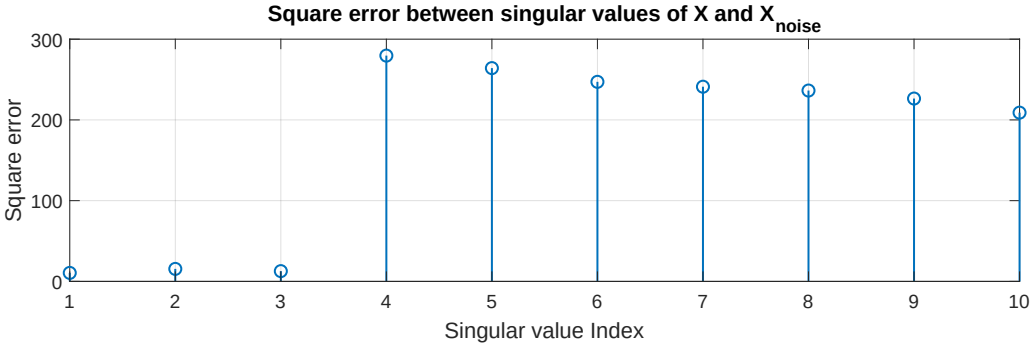


Figure 1.19: Square error between singular values of $\mathbf{X}$ and $\mathbf{X}_{noise}$

As Figure 1.18 shows, the number of non-zero singular values of matrix $\mathbf{X}$ is 3, which means that $\mathbf{X}$ is a rank 3 matrix. When matrix $\mathbf{X}$ is corrupted by zero-mean Gaussian noise, producing matrix $\mathbf{X}_{noise}$, the additive noise causes the zero singular values to be perturbed to non-zero values, while also causing a small increase in the magnitude of the three non-zero singular values of $\mathbf{X}$. The changes causes in the singular values is expressed by the square error between the singular values (Figure 1.19, where significant errors occur after the fourth singular value. The difference of the magnitude of singular values of the two matrices, can be thought of as increasing the variance of noise from zero to a non-zero value. The singular values of noise matrix $\mathbf{N}_X$, are related to noise variance by equation 1.12, which are the ones causing the increase of the zero valued singular values of $\mathbf{X}$.

$$\text{Variance } (\sigma^2) = \frac{1}{MN}\sum_{i=1}^{R} s_n^2(i) \tag{1.12}$$

where MxN are the dimensions of matrix $\mathbf{X}$, R is the number of variables and $s_n(i)$ is the i^{th} singular value of $\mathbf{N}_X$. In this case, the variance of noise is low (≈ 0.25), which allows singular values of the signal of interest to be separated in magnitude from those due to noise, which have much lower amplitude. However, when the signal-to-noise (SNR) ratio is very low, i.e. when variance of noise is high, then magnitudes of noise singular values will increase and there would be no significant difference between singular value amplitudes. Therefore, the rank of matrix $\mathbf{X}_{noise}$ would become hard to identify. It also worth noting that the slight increase in first three singular values of $\mathbf{X}_{noise}$ with increasing variance of noise, might be due to spurious correlations between signal and noise subspaces, or from the fact that the singular values of noise matrix $\mathbf{N}_x$ are not the same (ACF of noise is not exactly a delta function due to small sample size).

Part b)

The rank of matrix $\mathbf{X}_{noise}$ was determined to be 3, therefore the 3 most significant singular values (called principal components) were used to create a low-rank approximation of $\mathbf{X}_{noise}$, denoted by $\tilde{\mathbf{X}}_{noise}$. This is basically a "filtered" or de-noised version of $\mathbf{X}_{noise}$, since smaller singular values are consider to be due to noise. The difference (square error) between the variables of the noiseless input matrix, $\mathbf{X}$, and those in the noise corrupted matrix $\mathbf{X}_{noise}$ and denoised matrix $\tilde{\mathbf{X}}_{noise}$, is computed and shown in Figure 1.20. The plot shows that the denoised signal provides a better approximation of the original signal, since the square error is lower for all input variables. This means that most of the original signal information is captured by the first three singular values, while the rest belongs to the additive noise. The fact that error does not reduce to zero, is due to the fact that some information is lost by the truncated SVD. It can also be observed that error is not reduced by the same amount for each input variable. From further analysis, it was observed that the SNR was different for each variable, and variables 4 and 10, with lowest

reduction in error, had in fact the highest SNR's. This suggests that the higher the SNR for the input variable, the harder it is to remove the noise (overfitting of noise).

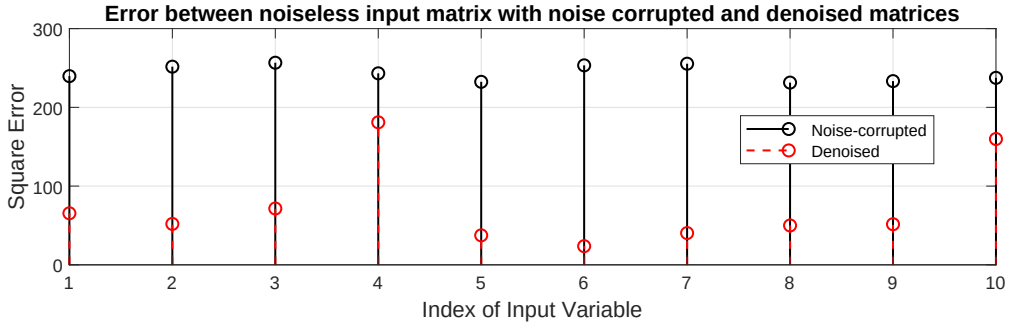


Figure 1.20: Difference (Square error) between the variables of the original noiseless matrix and those of noise corrupted and noiseless matrices.

Part c)

The Ordinary Least Square (OLS) and Principal Component regression (PCR) solutions for the unknown parameter matrix $\mathbf{B}$, which relates $\mathbf{X}_{noise}$ and $\mathbf{Y}$, are calculated, giving $\hat{\mathbf{B}}_{OLS}$ and $\hat{\mathbf{B}}_{PCR}$ respectively. Then, the estimation error (square error) between $\mathbf{Y}$ and $\hat{\mathbf{Y}}_{OLS} = \mathbf{X}_{noise}\hat{\mathbf{B}}_{OLS}$ and $\hat{\mathbf{Y}}_{PCR} = \mathbf{X}_{noise}\hat{\mathbf{B}}_{PCR}$ is computed for each variable of training data and results are summarised in Table 1. To avoid large numbers, the square error for each variable was normalised by the number of realisations of that variable, i.e. by a factor of 1000 for every variable.

Table 1: Normalised estimation error between OLS and PCR solutions for parameter matrix B, for each variable of training-set.

	Training-set Variable					
Method	**1**	**2**	**3**	**4**	**5**	**Total**
OLS	1.1609	0.6106	0.5727	0.3832	0.8243	3.5517
PCR	1.1682	0.6163	0.5788	0.3866	0.8272	3.5771

In order to validate the validate the effectiveness of the regression model derived from the OLS and PCR solutions, a test-set is used which is another realisation of the signal drawn from the statistical distribution of the training set. The data from the test-set were also estimated using the coefficients determined from the training set. The performance of the two methods was quantified by the square error between $\mathbf{Y}_{test}$ and $\hat{\mathbf{Y}}_{test-OLS} = \mathbf{X}_{test}\hat{\mathbf{B}}_{OLS}$ and $\hat{\mathbf{Y}}_{test-PCR} = \mathbf{X}_{test}\hat{\mathbf{B}}_{PCR}$. The results are again normalised by the same factor, equal to number of realisations for each variable, and are summarised in Table 2.

Table 2: Normalised estimation error between OLS and PCR solutions for parameter matrix B, for each variable of test-set.

	Test-set Variable					
Method	**1**	**2**	**3**	**4**	**5**	**Total**
OLS	8.3691	3.9699	3.2502	1.3983	5.6074	22.5950
PCR	8.3479	3.9552	3.2369	1.3990	5.6137	22.5528

The results from Tables 1 and 2 suggest that OLS methods performs better when the training data set is used, while PCR method performs better on the test data set. This is because the error of OLS method is lower than the error of PCR for all variables, while the opposite holds for the test-set. This can be explained from the fact that the OLS method over-fits the training set, since the OLS model is based on noise components as well. As a result, OLS model fits more closely the training set, but since it becomes tailored to fit the random noise in training data, it describes more weakly the underlying relationships between the variables, which results in a poorer performance in the test-set. On the other hand, PCR model is not affected by the noise in data, since PCR method denoises the input, resulting in lower complexity (no over-fitting), resulting in better performance in the test-set.

Part d)

To best way to validate the observations described in Part c) is to compare the effectiveness of the PCR solutions to the OLS solutions by testing the estimated regression coefficients from the two methods over an ensemble of test data. Using the provided MATLAB function `regval` a 1000 new realisations of the test data $\mathbf{Y}$, are obtained from which the estimate $\hat{\mathbf{Y}}$ is found using the regression coefficients from the two methods found in Part c). Based on the generated realisations of $\mathbf{Y}$ and $\hat{\mathbf{Y}}$, the mean square error (MSE) is estimated for the two methods given by MSE $= E\|\mathbf{Y} - \hat{\mathbf{Y}}\|_2^2$. Results are shown in Table 3, where to avoid large numbers, the MSE value was divided by the total number of realisations of each variable (i.e. factor of 1000). The results in Table 3 verify that PCR method outperforms the OLS method in modelling out-of-sample (test-set) data.

Table 3: Normalised MSE between OLS and PCR solutions for parameter matrix B, for each an ensemble of 500 realisations of test data.

	Test-set Variable					
Method	**1**	**2**	**3**	**4**	**5**	**Total**
OLS	0.9345	0.3933	0.3350	0.1361	0.5598	2.3587
PCR	0.9261	0.3884	0.3275	0.1328	0.5554	2.3301

2 Adaptive signal processing

2.1 The Least Mean Square (LMS) Algorithm

Part a)

In this section we are given that the process $x(n)$ is a second-order auto-regressive (AR) process satisfying the difference equation described by Equation 2.1.

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n) \ , \ where \ \eta(n) \sim \mathcal{N}(0, \sigma_\eta^2) \tag{2.1}$$

The original parameters in Equation 2.1 are $a_1 = 0.1$, $a_2 = 0.8$ and $\sigma_\eta^2 = 0.25$. Using the LMS estimator, we can approximate the AR coefficients (a_1,a_2) using the signal $x(n)$. This method is effectively using an LMS filter as a dynamical system with input vector $\mathbf{x}(n) = [x(n), x(n-1)]^T$. By considering the autocorrelation matrix of input vector ($\mathbf{R}_{xx}$), given by Equation 2.2, we can identify the range of values of step-size μ for which LMS converges in the mean to the Wiener-Hopf solution.

$$\begin{aligned}\mathbf{R}_{xx} = E\{\mathbf{x}(n)\mathbf{x}(n)^T\} = E\{[x(n), x(n-1)]^T[x(n), x(n-1)]\} = E\left\{\begin{bmatrix} x(n)x(n) & x(n)x(n-1) \\ x(n-1)x(n) & x(n-1)x(n-1)\end{bmatrix}\right\} \\ = \begin{bmatrix} E\{x(n)x(n)\} & E\{x(n)x(n-1)\} \\ E\{x(n-1)x(n)\} & E\{x(n-1)x(n-1)\}\end{bmatrix} = \begin{bmatrix} r_x(0) & r_x(1) \\ r_x(1) & r_x(0)\end{bmatrix}\end{aligned} \tag{2.2}$$

In Equation 2.2, we assume that the signal $x(n)$ is wide-sense stationary (WSS), where autocorrelation function (ACF), r_x, is only dependent on time-lag and we also utilised the symmetry property of ACF. To find the elements of matrix $\mathbf{R}_{xx}$, we need to multiply both sides of Equation 2.1 by $x(n-m)$, where m is an arbitrary time-lag, and then obtain the expected value.

$$\begin{aligned}E\{x(n)x(n-m)\} &= a_1 E\{x(n-1)x(n-m)\} + a_2 E\{x(n-2)x(n-m)\} + E\{\eta(n)x(n-m)\} \\ &= a_1 r_x(1-m) + a_2 r_x(2-m) + E\{\eta(n)x(n-m)\}\end{aligned} \tag{2.3}$$

The last term of Equation 2.3, will have a value of σ_η^2 when $m = 0$ and a value of 0 when $m \neq 0$. This is due to the fact that WGN is independent of past values of $x(n)$ and also $E\{\eta(n)x(n)\} = E\{\eta^2(n)\} = \sigma_\eta^2$. This can allow us to obtain a set of Yule-Walker equations (Equations 2.4) that relate the ACF of x(n) with the AR coefficients.

$$\begin{aligned}r_x(0) &= a_1 r_x(1) + a_2 r_x(2) + \sigma_\eta^2 \\ r_x(1) &= a_1 r_x(0) + a_2 r_x(1) \\ r_x(2) &= a_1 r_x(1) + a_2 r_x(0)\end{aligned} \tag{2.4}$$

By solving the set of Equations 2.4, we can find the values of $r_x(0)$ and $r_x(1)$, which are 25/27 and 25/54 respectively, and thus we obtain $\mathbf{R}_{xx}$, given below:

$$\mathbf{R}_{xx} = \begin{bmatrix} 25/27 & 25/54 \\ 25/54 & 25/27\end{bmatrix} \approx \begin{bmatrix} 0.926 & 0.463 \\ 0.463 & 0.926\end{bmatrix} \tag{2.5}$$

In order for the LMS to converge in the mean to the Wiener optimal solution, the step-size μ must be in the range $0 < \mu < 2/\lambda_{max}$, where λ_{max} is the largest eigenvalue of autocorrelation matrix $\mathbf{R}_{xx}$. The theoretical λ_{max} of the matrix $\mathbf{R}_{xx}$ given in Equation 2.5 is equal to 1.389, therefore this means $0 < \mu < 1.44$.

Part b)

The LMS adaptive predictor is implemented in MATLAB for the process $x(n)$ described by equation 2.1, using N=1000 samples. The performance of the algorithm was evaluated by obtaining its squared prediction error between actual model $x(n)$ and estimated model $\hat{x}(n)$, $e^2(n) = (x(n) - \hat{x}(n))^2$, in dB for step-sizes μ of 0.05 and 0.01. The same procedure was repeated for 100 realisations of $x(n)$, where the learning curve was obtained by averaging the squared prediction errors between the different realisations. Results are shown in Figure 2.1.

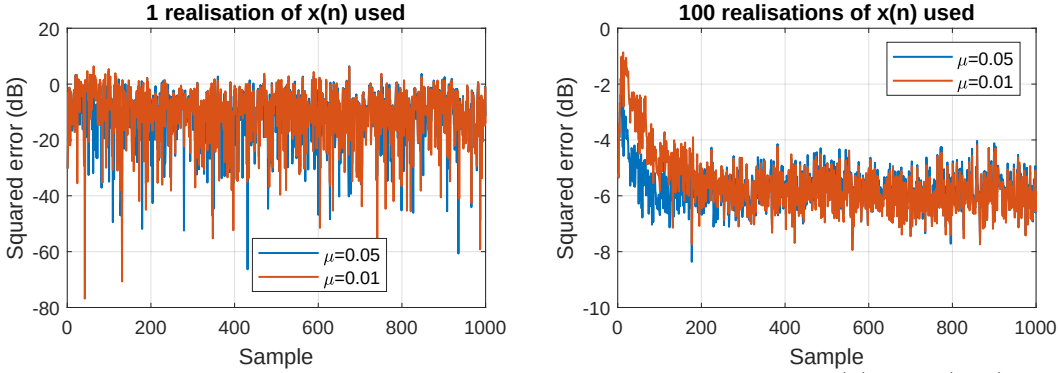


Figure 2.1: Plot of squared prediction error in dB of LMS adaptive predictor for process $x(n)$ using (Left) one realisation and (Right) 100 realisations.

Due to high variance of the squared error using a single realisation, we cannot reach to any conclusions about the convergence of the algorithm or the effect of different step-sizes. However, when 100 realisations were used and their squared error was averaged, we can observe a larger convergence speed for $\mu = 0.05$, which converges to a steady state

value of error at around 100 samples while for $\mu = 0.01$ error converges after 200 samples approximately. However, for step size of $\mu = 0.05$ the error converges to relatively higher steady state error than step-size $\mu = 0.01$.

Part c)

The LMS algorithm weight estimates fluctuate around the optimal solution (Wiener-Hopf solution) introducing excess error to the minimum achievable error signal. This additional error power is quantified by the excess mean square error (EMSE), which equals the difference between the MSE of LMS algorithm, where MSE= $\lim_{n\to\infty} E\{e^2(n)\}$, and the minimum attainable MSE of a Wiener filter, equal to σ_η^2. The misadjustment metric, $\mathcal{M}$, gives the ratio between EMSE and MSE of Wiener filter, i.e. $\mathcal{M} = EMSE/\sigma_\eta^2$, which for small step-sizes can be approximated as:

$$\mathcal{M}_{LMS} \approx \frac{\mu}{2} Tr\{\mathbf{R}_{xx}\} \tag{2.6}$$

By time-averaging over the steady-state of the ensemble-averaged learning curves (using 100 realisations) for the step-sizes $\mu = 0.05$ and $\mu = 0.01$, the estimated MSE error of the LMS algorithm was obtained, from which the misadjustment value was estimated. The estimated and theoretical LMS misadjustment values are shown on Table 4. For this calculation, it was estimated that the learning curves have reached their steady-state values after 600 samples for $\mu = 0.05$ and 800 samples for $\mu = 0.01$ (from Figure 2.1), although the transient period for $\mu = 0.05$ might in fact be shorter and also for $\mu = 0.01$ the transient period might not have ended.

Table 4: Estimated and theoretical LMS misadjustments. Theoretical misadjustments computed using Equation 2.6.

μ	$\mathcal{M}_{estimated}$	$\mathcal{M}_{theoretical}$
0.01	0.0106	0.0093
0.05	0.0513	0.0463

Table 4 shows that the estimated misadjustment values are very close to the theoretical values, where $\mathcal{M}$ is higher for larger μ since $\mathcal{M} \approx 0.5\mu N\sigma_x^2$. Specifically, the absolute error for $\mu = 0.01$ is 14.0% and for $\mu = 0.05$ is 10.8%. These deviations can be explained from the fact that the theoretical value of $\mathcal{M}$ is just an approximation, transient period length was also approximated, data length is not infinite (N=100) and also a finite ensemble of realisations was used. Although both step-sizes ensure convergence both in the mean and MSE (MSE convergence when $0 < \mu < 2/trace[\mathbf{R}_{xx}] = 1.08$)), the MSE error of LMS algorithm using $\mu = 0.05$ is higher, deviating more from MSE of Wiener filter and resulting in higher value of $\mathcal{M}$. Even though the algorithm converges faster using $\mu = 0.05$ to steady state, it oscillates with a larger steady-state error variance and the larger steps may cause the optimal solution to be missed. Therefore, the bias and variance of LMS coefficients are higher for larger μ values (but decrease to zero and σ_η^2 respectively as sample size goes to infinity) resulting in higher mean square weight error (misalignment), since $MSE = bias^2 + variance$ (Figure 2.2).

Part d)

By averaging the trajectories of the adaptive LMS filter parameters, from the 100 realisations of the process, the average time-evolution of coefficients is obtained for the step-sizes of $\mu = 0.05$ and $\mu = 0.01$ and are shown in Figure 2.3. By averaging the steady-state values of the coefficients over the 100 trials of process, we obtain an estimation of the true coefficients. Specifically, the estimated value ($\hat{a}_1$) for $a_1 = 0.1$ is 0.0977 for $\mu = 0.01$ and 0.0729 for $\mu = 0.05$. Also, the estimated value ($\hat{a}_2$) for $a_2 = 0.8$ is 0.766 for $\mu = 0.01$ and 0.715 for $\mu = 0.05$.

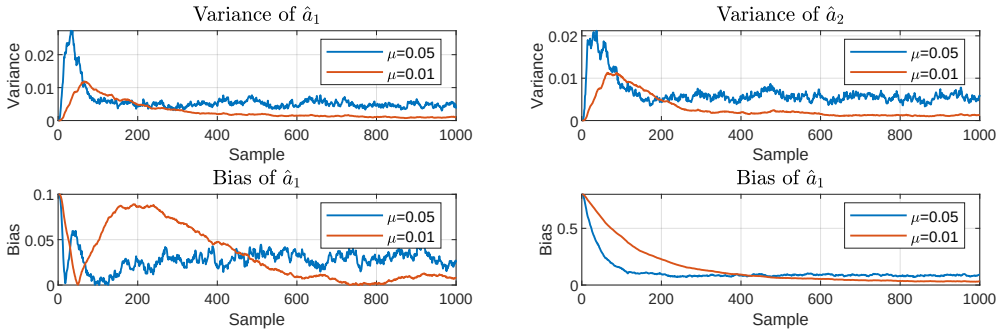


Figure 2.2: Plot of estimated Variance and Bias for each coefficient using $\mu = 0.05$ and $\mu = 0.01$, for 100 iterations.

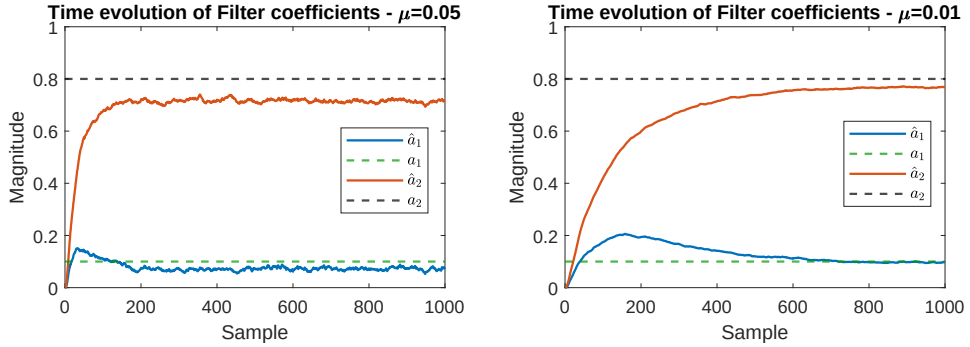


Figure 2.3: Time evolution of adaptive filter coefficients for $\mu = 0.05$ and $\mu = 0.01$.

As explained in part c), for larger μ values, the algorithm converges faster to the steady-state values, due to larger

time-steps taken, but oscillates with higher variance in the steady-state. The large time-steps, can cause optimal solutions to be missed, resulting in higher error in estimation compared to lower μ values. In general, there is a trade-off between convergence speed and convergence in MSE.

Part e)

Given that the cost function for the Leaky LMS algorithm is given by $J_2(n) = \frac{1}{2}(e^2(n) + \gamma\|w(n)\|_2^2)$, where $e(n) = d(n) - y(n) = d(n) - \mathbf{w}^T(n)\mathbf{x}(n)$ with $y(n)$ being the predicted system output and $d(n)$ is the desired system output, we can derive the update rule for Leaky LMS coefficients $\mathbf{w}(n)$ that minimises $J_2(n)$. Firstly, we know that the update equation for the steepest descent method to find the minimum of the cost function J_2 is given by:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\nabla_{\mathbf{w}}J_2(n) \tag{2.7}$$

Starting for the equation of $J_2(n)$, we can express it as:

$$J_2(n) = \frac{1}{2}(e^2(n) + \gamma\|w(n)\|_2^2 = \frac{1}{2}e^2(n) + \frac{\gamma}{2}\mathbf{w}^T(n)\mathbf{w}(n) \tag{2.8}$$

Therefore, the gradient of J_2 is given by:

$$\begin{aligned}\nabla_{\mathbf{w}}J_2(n) &= \frac{1}{2}\frac{\partial e^2(n)}{\partial e(n)}\frac{\partial e(n)}{\partial y(n)}\frac{\partial y(n)}{\partial \mathbf{w}(n)} + \frac{\gamma}{2}\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^T(n)\mathbf{w}(n)\right) = \frac{1}{2}(2e(n))(-1)(\mathbf{x}(n)) + \gamma\mathbf{w}(n) \\ &= \gamma\mathbf{w}(n) - e(n)\mathbf{x}(n)\end{aligned} \tag{2.9}$$

Substituting Equation 2.9 into Equation 2.7, gives:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\left[\gamma\mathbf{w}(n) - e(n)\mathbf{x}(n)\right] = \mathbf{w}(n)\left[1 - \gamma\mu\right] + \mu e(n)\mathbf{x}(n) \tag{2.10}$$

Therefore, this shows that the update rule of the Leaky LMS algorithm, described by Equation 2.10, is in the direction of steepest descent of the cost function J_2.

Part f)

The Leaky LMS algorithm was implemented in MATLAB, for different values of μ and γ, to estimate the AR coefficients of the signal described by 2.1. The estimated AR coefficients were obtained by averaging the steady-state values of the coefficients (obtained along the the final iterations of the Leaky-LMS) over 100 independent trials of the experiment. The results obtained are summarised in Table 5. The absolute error shown in Table 5, is given by $(|a_i - \hat{a}_i|/a_i) \times 100\%$.

Table 5: Estimated and true coefficients of Leaky-LMS filter for different μ and γ values, with the corresponding absolute error.

$\boldsymbol{\mu}$	$\boldsymbol{\gamma}$	$\mathbf{a_1}$	$\hat{\mathbf{a}}_1$	**Absolute Error (%)**	$\mathbf{a_2}$	$\hat{\mathbf{a}}_2$	**Absolute Error (%)**
0.01	**0**	**0.1**	0.1008	0.8	**0.8**	0.763	4.7
0.05	**0**	**0.1**	0.0732	26.8	**0.8**	0.7099	11.3
0.01	**0.2**	**0.1**	0.128	27.5	**0.8**	0.609	23.8
0.05	**0.2**	**0.1**	0.0914	8.6	**0.8**	0.543	32.1
0.01	**0.5**	**0.1**	0.143	43.4	**0.8**	0.464	42.1
0.05	**0.5**	**0.1**	0.108	8.1	**0.8**	0.405	49.4
0.01	**0.8**	**0.1**	0.139	38.7	**0.8**	0.382	52.3
0.05	**0.8**	**0.1**	0.0988	1.2	**0.8**	0.326	59.2

The Leaky-LMS algorithm allows stability of the LMS algorithm when the autocorrelation matrix of the process has zero eigenvalues, by forcing filter weights to converge (increases the eigenvalues of autocorrelation matrix). In the case of zero eigenvalues, the autocorrelation matrix $\mathbf{R}_{xx}$ shifts from positive definite to positive semi-definite, thus invertibility is not guaranteed. However, the weights of the Leaky-LMS algorithm do not converge to the true coefficients of the process. In fact, the higher the value of the leakage coefficient γ, the higher the deviation (bias) from the true AR coefficients, as shown in the Table 5. This effect can be explained from the fact that the optimal Leaky-LMS weights are equal to $w_{optLeaky} = (\mathbf{R}_{xx} + \gamma\mathbf{I})^{-1}\mathbf{p}$, compared to the Wiener-Hopf solution where $w_{opt} = \mathbf{R}_{xx}^{-1}\mathbf{p}$, with $\mathbf{p}$ being the cross-correlation vector between input vector $\mathbf{x}(n)$ and desired output d(n), with estimates shown in Table 6. This means that optimal Leaky-LMS weights converge to the Wiener-Hopf solution in the mean as γ goes to zero. Similarly, the MSE of the Leaky-LMS algorithm does not converge to the minimum attainable MSE of a Wiener filter (both variance and bias of estimates increases). Additionally, we can see that from equation 2.10, the leakage coefficient γ reduces the sensitivity of algorithm coefficient updates to current coefficient values.

Table 6: Estimated optimal coefficient values using 1000 iterations of process to estimate the cross-correlation vector $\mathbf{p}(n)$.

$\boldsymbol{\gamma}$	**0**	**0.2**	**0.5**	**0.8**
$\mathbf{a_{1opt}}$	0.102	0.130	0.146	0.142
$\mathbf{a_{2opt}}$	0.801	0.608	0.475	0.394

Interestingly, the magnitude of coefficient a_2 is more affected by γ compared to a_1, since by rounding up estimates, $\hat{a}_1 \approx 0.1$ while $\hat{a}_2 \neq 0.8$. A possible reason for this is that the magnitude of $a_2 > a_1$.

2.2 Adaptive Step Sizes

Part a)

The three gradient adaptive step-size (GASS) algorithms, introduced by Benveniste, Ang & Fang, and Matthews & Xie, which aim to minimise the cost function the cost function with respect to the weight vector and also with respect to the step-size, were implemented in MATLAB. Their performance was compared when identifying a real-valued MA(1) system given by:

$$x(n) = 0.9\eta(n-1) + \eta(n) \text{ ,where } \eta(n) \sim \mathcal{N}(0, 0.5) \tag{2.11}$$

Two methods were implemented for modelling this process. **Method 1**, used initially, involved forcing the coefficient of $\eta(n)$ to be 1, estimate $w_0 = 0.9$ and use $\eta(n)$ only in error e(n) calculation, such that $e(n) = x(n) - (\hat{w}_0(n)\eta(n-1) + \eta(n))$. Despite the method being successful in identifying the MA(1) system, it had the disadvantage of being a noiseless regression problem (idealised case) that resulted in no fluctuations (i.e. 0 steady state error for all stable algorithms) since effectively $e(n) = w_0\eta(n-1)+\eta(n)) - (\hat{w}_0(n)\eta(n-1)+\eta(n)) = \eta(n-1)(w_0 - \hat{w}_0(n))$. Additionally for this method it was noticed that for standard LMS, if μ increases (e.g. to a value of 1) while convergence speed increases, steady-state error decreases, which is not valid (Figure 2.4). This led to **Method 2**, which was used for all MA process-modelling tasks, where only $\eta(n-1)$ was used in error e(n) calculation, such that $e(n) = x(n) - (\hat{w}_0(n)\eta(n-1))$, which introduces some uncertainty $\eta(n)$ in the prediction, which is found in real-signals. This method successfully illustrates the trade-off between convergence speed and steady-state error. The performance of GASS algorithms, using **Method 2**, was also compared to the standard LMS with fixed step-size (for $\mu = 0.01$ and $\mu = 0.1$). To allow comparisons, the weight error curves and squared weight error curves (in dB) were plotted, shown in Figure 2.4.

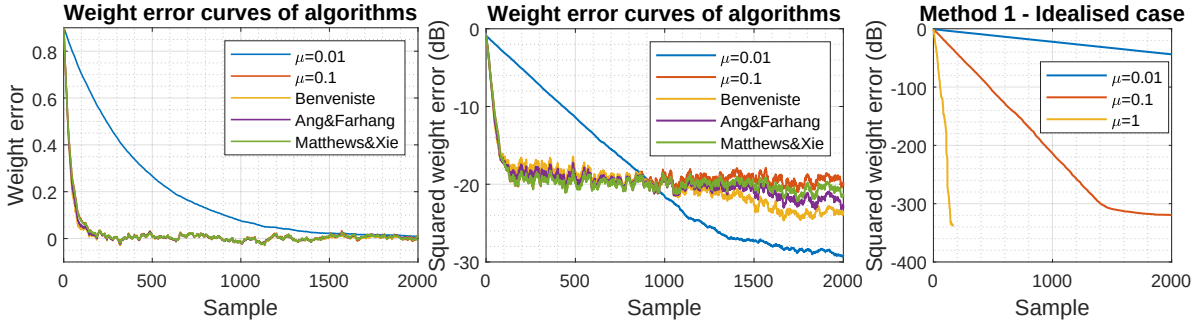


Figure 2.4: (Left) Weight error ($\tilde{w}(n) = 0.9 - w(n)$) curves for the three GASS algorithms and standard LMS algorithm with fixed step-size of $\mu = 0.01$ and $\mu = 0.1$, using **Method 2**. For all GASS algorithms initial step-size was $\mu = 0.1$, learning rate step-size $\rho = 0.005$ and parameter for Ang&Farhang algorithm was $\alpha = 0.9$. (Middle) Squared weight error curves in dB with **Method 2**. (Right) Squared weight error curves in dB using different μ for LMS, using **Method 1**, indicating no trade-off.

In general, the standard LMS algorithm suffers from a trade-off between convergence speed and steady-state error variance, which makes choice of fixed step-size μ challenging. Reaching convergence for LMS also dependents on data size available. However, the standard LMS algorithm is very fast to compute due to its low complexity, compared to GASS algorithms. On the other hand, the GASS algorithms allow a variable step-size to be used, based on the minimisation of the cost function with respect to step-size μ. Therefore, these allow a large step-size to be used initially to allow fast convergence and and small step-size as the parameter estimates approach closer to the true values, thus reducing the steady-state error. As shown in Figure 2.4, the GASS algorithms, especially Benveniste and Ang&Farhang (yellow and purple plots in Left Figure 2.4), converge to the true value of coefficient slightly faster than Matthews & Xie and LMS with $\mu = 0.1$, indicating the high convergence speed, and also have the minimum steady-state squared weight error.

The Beneviste algorithm has the best performance as suggested by Figure 2.4, converging to the true solution with the highest convergence speed and lowest weight squared error. The algorithm with second best performance is found to be Ang & Farhang, which however can outperform Benveniste's algorithm if $\alpha \approx 0.99$. At the same time though the initial step-size of Benveniste can be increased to outperform Ang & Farhang algorithm. When the optimal values of α and μ_0 are used, Benveniste's algorithm performs better (higher convergence speed and lower steady-state error) than Ang & Farhang algorithm, while error until convergence is higher for Benveniste. The third GASS algorithm by Matthews & Xie, has similar performance to the standard LMS algorithm with step-size $\mu = 0.1$, but with lower steady-state error. Finally, the standard LMS algorithm with step-size $\mu = 0.01$ has the poorest performance due to its high square error and its low convergence speed, but after convergence has the lowest steady-state error. The steady-state square weight errors after 2000 samples, where all algorithms have converged, are -23.6dB for Benveniste, -23.1dB for Ang & Farhang, -21.3dB for Matthews & Xie, -19.8dB for $\mu = 0.1$ and -29.2dB for $\mu = 0.01$. The GASS algorithms are generally more robust than standard LMS algorithm in terms of error and provide faster convergence, in the expense of higher computational complexity. The GASS lagorithms aim to minimise $\nabla_\mu J$ using the learning rate update rule $\mu(n+1) = \mu(n) + \rho e(n)\mathbf{x}^T(n)\boldsymbol{\psi}(n)$, where each algorithm set parameter $\boldsymbol{\psi}(n)$ (sensitivity) differently. Specifically, the Benveniste algorithm, which has the highest computational complexity in terms of learning rate update equation ($\mathcal{O}(M^2)$ for filter order M) due to matrix multiplications involved, its sensitivity $\boldsymbol{\psi}(n)$ involves a filtering term acting as a time-varying adaptive low-pass filter, which reduces (smooths) noise in instantaneous gradient estimates and makes algorithm very accurate and more robust. Additionally, the Belveniste's algorithm accuracy arises from the fact that is derived exactly through the minimisation of cost error and does not consider any independence assumptions in its form. For Ang & Farhang algorithm ($\mathcal{O}(M)$), the filtering term in sensitivity $\boldsymbol{\psi}(n)$ is replaced by lowpass filter

with a fixed coefficient α, thus reducing complexity compared to Benveniste. Finally, Matthews & Xie algorithm is a simplification to Ang & Farhang algorithm, where the coefficient $\alpha = 0$, reducing further the complexity (although still $\mathcal{O}(M)$) with the expense of further degradation to its performance (very close to performance of $\mu = 0.01$), since sensitivity is based on noisy gradient estimates.

Part b)

In order to verify that the update equation of LMS algorithm

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p \mathbf{x}(n) \tag{2.12}$$

based on the *a posteriori error* $e_p = d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1)$ is equivalent to the Normalised LMS (NLMS) algorithm, we start by expressing $\Delta\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n)$ in terms the *a priori* error $e(n)$. The *a priori* error is expressed as $e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n)$. Starting from Equation 2.12, multiply first both sides by $\mathbf{x}(n)^T$.

$$\mathbf{x}^T\mathbf{w}(n+1) = \mathbf{x}^T\mathbf{w}(n) + \mu e_p \mathbf{x}^T\mathbf{x}(n) = \mathbf{x}^T\mathbf{w}(n) + \mu e_p \|\mathbf{x}(n)\|^2 \tag{2.13}$$

We can then multiply both sides of 2.13 by a factor of -1 and then add d(n) to both sides.

$$\begin{aligned} d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1) &= d(n) - \mathbf{x}^T(n)\mathbf{w}(n) - \mu e_p \|\mathbf{x}(n)\|^2 \\ e_p(n) &= e(n) - \mu e_p \|\mathbf{x}(n)\|^2 \quad \rightarrow \quad e(n) = e_p(n)\left[1 + \mu\|\mathbf{x}(n)\|^2\right] \\ e_p(n) &= e(n)\left[\frac{1}{1+\mu\|\mathbf{x}(n)\|^2}\right] = e(n)\left[\frac{1+\mu\|\mathbf{x}(n)\|^2 - \mu\|\mathbf{x}(n)\|^2}{1+\mu\|\mathbf{x}(n)\|^2}\right] = e(n)\left[1 - \frac{\mu\|\mathbf{x}(n)\|^2}{1+\mu\|\mathbf{x}(n)\|^2}\right] \end{aligned} \tag{2.14}$$

By substituting Equation 2.14 into $\Delta\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n) = \mu e_p \mathbf{x}(n)$ we obtain Equation 30 provided in coursework description.

$$\Delta\mathbf{w}(n) = \mu e(n)\left[1 - \frac{\mu\|\mathbf{x}(n)\|^2}{1+\mu\|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) \tag{2.15}$$

This can be further simplified to bring it to a form that will allow direct comparisons with the NLMS algorithm.

$$\begin{aligned} \Delta\mathbf{w}(n) &= e(n)\left[\mu - \frac{\mu^2\|\mathbf{x}(n)\|^2}{1+\mu\|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) = e(n)\left[\mu - \frac{\mu\|\mathbf{x}(n)\|^2}{\frac{1}{\mu}+\|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) \\ \Delta\mathbf{w}(n) &= e(n)\left[\frac{1+\mu\|\mathbf{x}(n)\|^2 - \mu\|\mathbf{x}(n)\|^2}{\frac{1}{\mu}+\|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) = e(n)\left[\frac{1}{\frac{1}{\mu}+\|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) \end{aligned} \tag{2.16}$$

Since the weight update for NLMS is given by:

$$\Delta\mathbf{w}_{NMLS}(n) = e(n)\left[\frac{\beta}{\epsilon + \|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) \tag{2.17}$$

by comparing Equation 2.17 with Equation 2.16, we can show that weight update equation based on the *a posteriori error* is equivalent to the NLMS update equation with $\beta = 1$ and $\epsilon = 1/\mu$.

Part c)

The Generalised Normalised Gradient Descent (GNGD) Algorithm was implemented in MATLAB for identification of the process described by Equation 2.11, using **Method 2** described in Part a). The GNGD algorithm improves stability of the NLMS algorithm by making the generalisation factor ϵ time-varying via a gradient method given by:

$$\epsilon(n+1) = \epsilon(n) - \rho\mu\frac{e(n)e(n-1)\mathbf{x}^T(n)\mathbf{x}(n-1)}{\left(\epsilon(n-1) + \|\mathbf{x}(n-1)\|^2\right)^2} \tag{2.18}$$

The weight update equation of the GNGD algorithm is given by Equation 2.17. For the implemented algorithm the parameters were set to $\beta = 1$, $\rho = 0.005$ and $\epsilon(n=0) = 1/\mu$. The performance of GNGD was then compared to the Benveniste's algorithm by considering the evolution weight estimates for the two methods, illustrated in Figure 2.5.

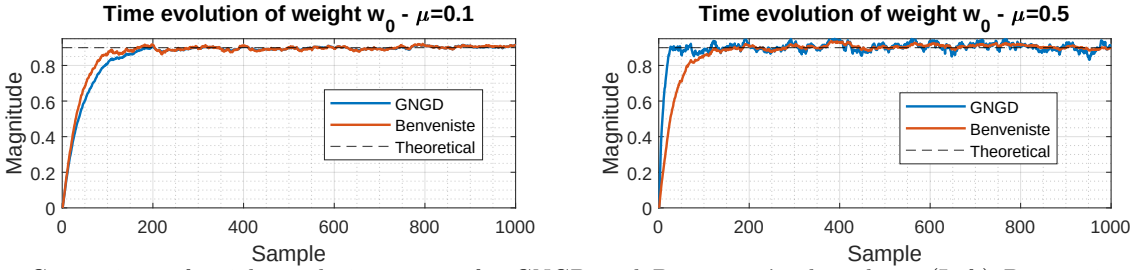


Figure 2.5: Comparison of weight evolution curves for GNGD and Benveniste's algorithm. (Left) Done using $\mu_{GNGD} = \mu_{benveniste}(0) = 0.1$ and $\rho = 0.005$. (Right) Done using $\mu_{GNGD} = 0.5$, $\mu_{benveniste}(0) = 0.1$ and $\rho = 0.005$.

As shown in Figure 2.5, the GNGD algorithm is able to converge to the optimal solution for a range of μ values, although the convergence speed of algorithm is highly affected by μ. In the case that both μ_{GNGD} and $\mu(0)_{Benveniste}$ have the same value (for low values of $\mu \leq 0.1$), then Benveniste's algorithm performs better than the GNGD algorithm (faster convergence). In fact, for $\mu < 0.02$ GNGD algorithm does not converge to optimal solution after 1000 samples. However, when parameter μ_{GNGD} increases while $\mu(0)_{Benveniste}$ stays the same, then the GNGD algorithm can perform better and converge faster than Benveniste's algorithm, but with higher steady-state error. It was also noticed that using the idealised zero-steady-steady error **Method 1**, described in Part a), that even in cases for large μ_{GNGD} values (> 2), the GNGD algorithm is still converging (stable) while Benveniste's algorithm for these high $\mu(0)_{Benveniste}$ values becomes unstable (diverges) (Figure 2.6). This difference is due to the fact that the

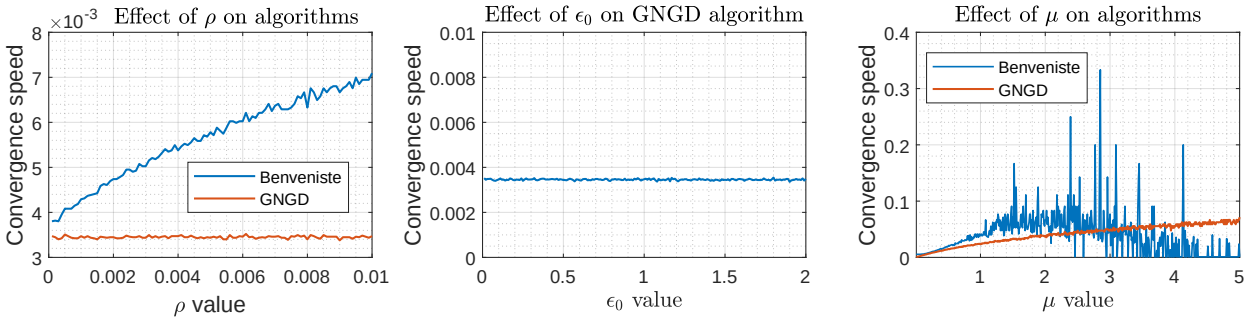


Figure 2.6: (Left) Investigating the effect of ρ on the performance of both algorithms ($\mu_{GNGD} = \mu_{benveniste}(0) = 0.1$). (Middle) Investigating the effect of ϵ_0 on the performance of GNGD algorithm ($\rho = 0.005, \mu_{GNGD} = 0.1$). (Right) Investigating convergence speed of Benveniste using $\mu_{benve}(0) = \mu$ and GNGD using $\mu_{GNGD} = \mu$. Convergence speed was used as performance metric, where convergence is considered when $|w_0 - \tilde{w}_0| < 0.001$, otherwise speed is set to 0. **Method 1** is used for the plots.

adaptive learning rate of GNGD is based on a non-linear update, while for Benveniste's algorithm updates its adaptive learning rate in a linear manner, which makes GNDG more robust (increased stability) to changes in initialisation of its parameters. As shown in Figure 2.6, we can empirically show that the performance of GNGD algorithm is also unaffected by the values of ρ and ϵ_0, while performance of Benveniste's algorithm increases with increasing ρ values. The computational complexity of the two algorithms has also been considered by estimating the number of additions (subtractions) and multiplications (divisions) of the two algorithms, which is illustrated in Tables 7 and 8. In the two tables, N_w denotes the number of components in weight vector $\mathbf{w}$, which is also equal to $N_w = M + 1$, where M is the model order (filter length). From these calculations we can observe that the complexity of GNGD algorithm grows linearly with model order (or N_w) while the complexity of Belveniste's algorithm grows quadratically with model order. Therefore, this means that for the same model order (and especially large model orders), the computational complexity of Belveniste's algorithm is much higher than that of GNGD algorithm. This is mainly due to the outer product of $\mathbf{x}(n-1)\mathbf{x}^T(n-1)$ involved in the update of the sensitivity vector $\boldsymbol{\psi}(n)$.

Table 7: Computational Complexity for Benveniste's algorithm for every time instant n. N_w is the number of components of weight vector $\mathbf{w}$, which is equal to $N_w = M + 1$, where M is the model order. Also for calculations, subtraction and division are considered as single-step operations.

Algorithm Step	**Sub-step**	**Multiplications/Divisions**	**Additions/Subtractions**
$\boldsymbol{\psi}(n)$	$A = e(n-1)\mathbf{x}(n-1)$	N_w	0
	$B = \mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)$	$2N_w^2$	$N_w(N_w-1)$
	$C = (\mathbf{I} - B)\boldsymbol{\psi}(n-1)$	N_w^2	$2N_w^2 - N_w$
	$C + A$	0	N_w
e(n)	$D = \mathbf{w}^T(n)\mathbf{x}(n)$	N_w	$N_w - 1$
	d(n) - D	1	1
$\mu(n+1)$	$E = \mathbf{x}^T(n)\boldsymbol{\psi}(n)$	N_w	$N_w - 1$
	$\mu(n) + \rho e(n)E$	2	1
$\mathbf{w}(n+1)$	$\mathbf{w}(n) + \mu(n)e(n)\mathbf{x}(n)$	$N_w + 1$	N_w
Total Number of computations		$3N_w^2 + 4N_w + 3$	$3N_w^2 + 2N_w$
Overall computational complexity		$\mathcal{O}(N_w^2)$	

Table 8: Computational Complexity for GNGD algorithm for every time instant n. N_w is the number of components of weight vector $\mathbf{w}$, which is equal to $N_w = M + 1$, where M is the model order. Also for calculations, subtraction and division are considered as single-step operations.

Algorithm Step	**Sub-step**	**Multiplications/Divisions**	**Additions/Subtractions**
$\mathbf{w}(n+1)$	$A = \|\mathbf{x}(n)\|^2$	N_w	$N_w - 1$
	$B = \frac{\beta}{\epsilon(n)+A}e(n)$	2	1
	$\mathbf{w}(n) + B\mathbf{x}(n)$	N_w	N_w
$\epsilon(n+1)$	$C = \|\mathbf{x}(n-1)\|^2$	N_w	$N_w - 1$
	$D = \mathbf{x}(n)^T\mathbf{x}(n-1)$	N_w	$N_w - 1$
	$e(n) - \rho\mu\frac{e(n)e(n-1)D}{(\epsilon(n-1)+C)^2}$	9	3
Total Number of computations		$4N_w + 11$	$4N_w + 3$
Overall computational complexity		$\mathcal{O}(N_w)$	

2.3 Adaptive Noise Cancellation

Part a)

In Adaptive Line Enhancer (ALE) a delay Δ is introduced in the input signal $s(n) = x(n) + \eta(n)$, where $x(n)$ is a clean sinusoidal signal ($x(n) = sin(0.01\pi n)$) and noise (coloured noise) term $\eta(n) = v(n) + 0.5v(n-1)$ with white noise $v(n) \sim \mathcal{N}(0, 1)$. Using an appropriate value of Δ, the input to the linear predictor component of ALE ($\mathbf{u}(n)$) will be uncorrelated to the noise ($\eta(n)$) in the noise-corrupted signal ($s(n)$), such that the noise is suppressed by the predictor

and an estimate of the denoised signal is obtained by $\hat{\mathbf{u}}(n) = \mathbf{w}^T(n)\mathbf{u}(n)$, where $\mathbf{u}(n) = [s(n-\Delta), ..., s(n-\Delta-M+1)]^T$. The minimum value of delay Δ to be used by the ALE configuration is the value of Δ which minimises the MSE between the noise-corrupted signal $s(n)$ and denoised signal $\hat{x}(n)$. Ideally, the minimum value of MSE will be equal to the noise power, i.e. $MSE_{min} = \mathbb{E}\{\eta(n)^2\}$, since ideally $\hat{x}(n) = x(n)$ so $s(n) - \hat{x}(n) = \eta(n)$.

$$\begin{aligned} MSE &= \mathbb{E}\{(s(n) - \hat{x}(n))^2\} = \mathbb{E}\{(x(n) + \eta(n) - \hat{x}(n))^2\} \\ &= \mathbb{E}\{(x(n) - \hat{x}(n))^2\} + 2\mathbb{E}\{(x(n) - \hat{x}(n))\eta(n)\} + \mathbb{E}\{\eta(n)^2\} \end{aligned} \tag{2.19}$$

The first term of Equation 2.19 ideally should be equal to zero and is irrelevant to noise $\eta(n)$. Since the minimum value of MSE is $\mathbb{E}\{\eta(n)^2\}$, this means that the second term in Equation 2.19 must be also zero, when value of Δ is appropriately chosen. By expanding the second term of Equation 2.19:

$$\mathbb{E}\{(x(n) - \hat{x}(n))\eta(n)\} = \mathbb{E}\{x(n)\eta(n)\} - \mathbb{E}\{\hat{x}(n)\eta(n)\} \tag{2.20}$$

Under the assumption that $x(n)$ is independent to zero-mean noise $\eta(n)$, $\mathbb{E}\{x(n)\eta(n)\} = \mathbb{E}\{x(n)\}\mathbb{E}\{\eta(n)\} = 0$. Therefore, Equation 2.20 reduces to:

$$\begin{aligned} \mathbb{E}\{(x(n) - \hat{x}(n))\eta(n)\} &= \mathbb{E}\{\hat{x}(n))\eta(n)\} = \mathbb{E}\{(v(n) + 0.5v(n-2))\mathbf{w}^T\mathbf{u}(n)\} = \mathbb{E}\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M} w_i u_i(n)\} \\ &= \mathbb{E}\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M} w_i s(n-\Delta-i)\} \\ &= \mathbb{E}\{(v(n) + 0.5v(n-2))\sum_{i=0}^{M} w_i (x(n-\Delta-i) + \eta(n-\Delta-i)\} \\ &= \sum_{i=0}^{M} w_i \left[\mathbb{E}\{(v(n) + 0.5v(n-2))x(n-\Delta-i)\} + \mathbb{E}\{(v(n) + 0.5v(n))\eta(n-\Delta-i)\}\right] \end{aligned} \tag{2.21}$$

Since $\eta(n) = v(n) + 0.5v(n-2)$ is also uncorrelated to $x(n-\Delta-i)$, Equation 2.21 further reduces to:

$$\begin{aligned} \mathbb{E}\{(x(n) - \hat{x}(n))\eta(n)\} &= \sum_{i=0}^{M} w_i \mathbb{E}\{(v(n) + 0.5v(n))\eta(n-\Delta-i)\} \\ &= \sum_{i=0}^{M} w_i \mathbb{E}\{(v(n) + 0.5v(n))(v(n-\Delta-i) + 0.5v(n-\Delta-2-i)\} \end{aligned} \tag{2.22}$$

Since $v(n)$ is i.i.d. white noise, $\mathbb{E}\{v(n)v(n-m)\} = 0$ for $m \neq 0$, therefore in order for Equation 2.22 to be zero, $\Delta > 2$. Thus, for a minimum $\Delta_{min} = 3$, the MSE is minimised. To verify empirically the theoretical minimum Δ value, 100 iterations of the noise-corrupted signal $s(n)$ and ALE estimate $\hat{x}(n)$, together with clean signal $x(n)$, are shown in Figure 2.7 for different Δ values, for filter order M=5 N=1000 data samples and $\mu_{LMS} = 0.01$. The minimun square prediction error (MSPE) being equal to $MSPE = (1/N)\sum_{n=0}^{N-1}(x(n) - \hat{x}(n))^2$, is also shown for each case and we observe that it decreases largely for $\Delta > 2$.

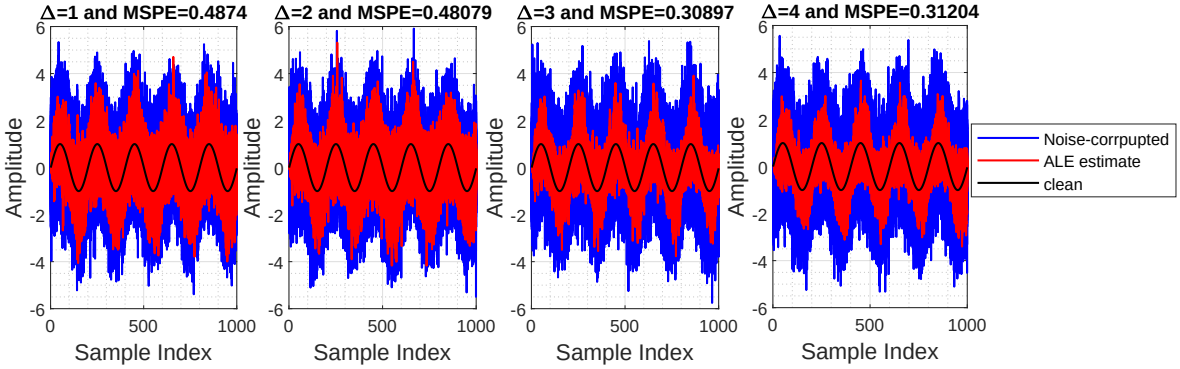


Figure 2.7: ALE implemented for 100 realisations of $s(n)$ for $\Delta = [1, 2, 3, 4]$ using model order 5.

Part b)

The same procedure is repeated by generating 100 realisations of the 1000-sample noise-corrupted signal $s(n)$, varying both the delay Δ and filter order M, while $\mu_{LMS} = 0.01$ (set empirically to this value to allow sufficient convergence and low steady-state variance). The results are illustrated in Figures 2.8 and 2.9. Observing the relationship between delay parameter Δ and MSPE in Figure 2.8 for model orders of M=[5,10,15,20], we can observe that for $\Delta \leq 2$ the MSPE for all model orders has a high value which decreases a Δ reaches value of 3, and MSPE value remains relatively flat until a value of $\Delta = 6$. However, as Δ increases beyond the value of 6, there is a steady increase in MSPE value for all values of M. This is because as Δ increases, there is increased delay (phase shift) introduced in ALE filter output, creating more pronounced deviations of denoised signal $\hat{x}(n)$ from clean signal $x(n)$. It should be noted that the higher the MSPE value, the poorer the performance of the ALE algorithm. The values of Δ for which the minimum value of MSPE occurs for each model order are shown in Table 9.

Table 9: Values of Δ for which minimum value of MSPE occurs for this experiment.

Model order (M)	5	10	15	20
Δ for MSPE$_{\text{min}}$	3	5	6	5

The performance (MSPE) of the de-noising algorithm is also affected by the filter order M, as shown in Figure 2.9. This shows that for low M values, the algorithm has high error due to the low degrees of freedom to sufficiently capture

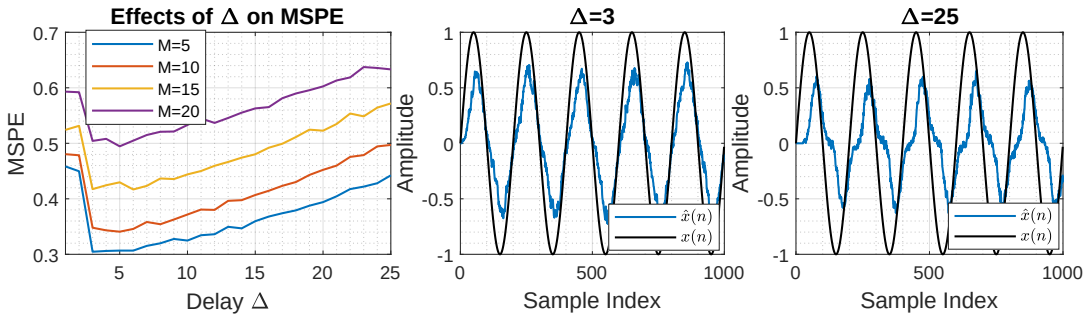


Figure 2.8: (Left) Effect of Δ on MSPE value for different model orders M. (Middle) Comparison of de-noised signal $\hat{x}(n)$ and clean signal $x(n)$ for $\Delta = 3$ and M=5. (Right) Comparison of de-noised signal $\hat{x}(n)$ and clean signal $x(n)$ for $\Delta = 25$ and M=5. The (Middle) and (Right) plots illustrate the phase shift introduced by ALE algorithm.

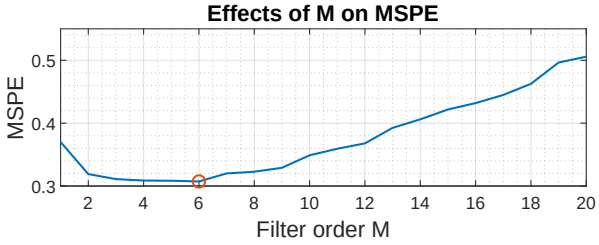


Figure 2.9: Relationship between filter order M and MSPE ($\Delta = 3$).

variations due to noise (under-fitting) and decreases to a minimum value as M increases to a value of 5. For higher model orders, the MSPE increases. This is because as filter order increases algorithm over-fits the noise in signal by increasing complexity, resulting in higher error and thus degradation of its performance.

Since the computational cost increases with M, a good pragmatic choice of order M should achieve low MSPE (ideally minimum) while M is kept at lowest value possible. As Figure 2.9 suggests, the minimum MSPE occurs at M=6 (MSPE = 0.30729), however for M=4 MSPE value (0.30872) is very close to minimum value (0.5% deviation). Therefore a good pragmatic choice of order is M=4. Additionally, the optimal value of delay parameter for this model order is found to be $\Delta = 3$.

Part c)

The adaptive noise cancellation (ANC) configuration was implemented in MATLAB and its performance was compared to that of the ALE configuration in de-noising the noise-corrupted sinusoid from part a). The MSPE measure was used as a metric for the relative performance of the two algorithms. The secondary coloured noise $\epsilon(n)$ given by Equation 2.23 was used as an input to the ANC linear predictor, which is linearly correlated to the coloured noise $\eta(n)$ found in the noisy signal $s(n)$. The relationship between $\epsilon(n)$ and $\eta(n)$ is unknown to the ANC algorithm.

$$\epsilon(n) = 0.7\eta(n) + 0.1 \tag{2.23}$$

Both algorithms were implemented using $\mu_{LMS} = 0.01$ and filter order M=5. For ALE algorithm the optimal delay of $\Delta = 3$ for M=5 was used. The results are shown in Figure 2.10.

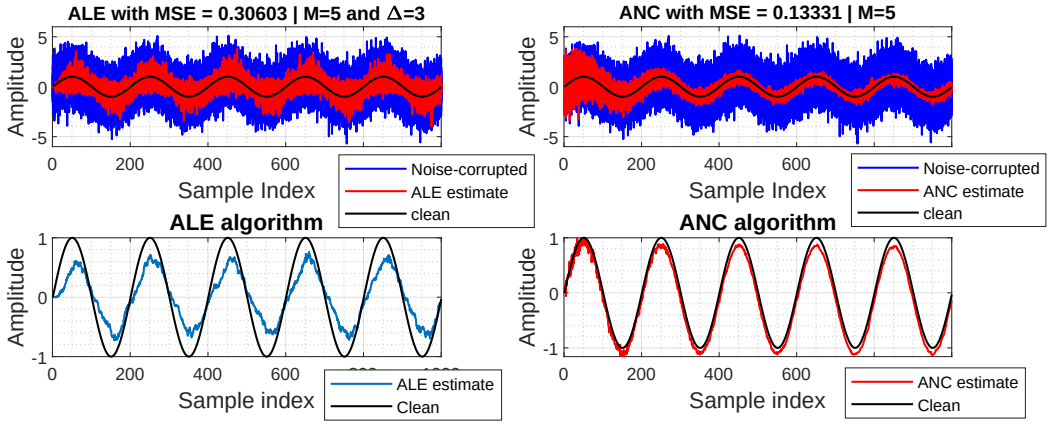


Figure 2.10: (Top) Overlay of 100 realisations of noise-corrupted signal $s(n)$ and de-noised signal $\hat{x}(n)$ for ALE and ANC algorithms. (Bottom) Ensemble-averaged de-noised signals for ALE and ANC algorithms compared to clean signal $x(n)$.

As Figure 2.10 shows, the ANC algorithm outperforms the ALE algorithm in de-noising the noise-corrupted sinusoid, since $MSPE_{ANC} = 0.133$ while $MSPE_{ALE} = 0.306$. However, it is worth noting that for sample index values $n < 400$, the ANC algorithm performs poorly resulting in large errors. These errors are minimised once the algorithm converges, allowing de-noised signal to track closely the clean signal with low steady-state errors. This can be explained from the fact that the performance of ALE is relatively constant with time whereas for ANC the algorithm performs poorly initially but performance increases as algorithm converges. This suggests that ALE performs better for small sample size data, while ANC performs better for sufficiently large sample sizes. Additionally, the performance of ANC algorithm is dependent on the choice of the secondary noise signal, which must be sufficiently (linearly) correlated to the noise in original signal for successful performance.

Part d)

In this part, the ANC method was applied on EEG recordings from the Cz electrode on the scalp, which as seen in Section 1.2 b), are corrupted by a strong 50Hz noise component introduced by the mains. Figure 2.11 shows the spectrogram of the EEG data using the `spectrogram` function in MATLAB. Each segment of periodogram plot was obtained using a Hanning window of length 6000 ($5 \times f_s = 5(1200)$) and 50% overlap between segments.

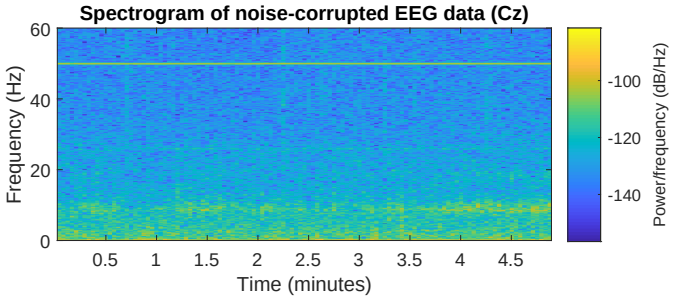


Figure 2.11: Spectrogram of original noise-corrupted EEG data from the Cz electrode.

The filter order M of LMS filter order and the step-size μ are varied in order to empirically determine the parameter values for which the 50Hz component in EEG data is suppressed without affecting the other frequency components. Additionally, a synthetic reference input signal $\epsilon(n)$ for ANC configuration is used composed of a unit amplitude sinusoid of 50Hz corrupted by zero-mean white Gaussian noise with 0.005 variance, which is correlated to the mains interference. The spectrogram plots for varying M and μ values are shown in Figure 2.12.

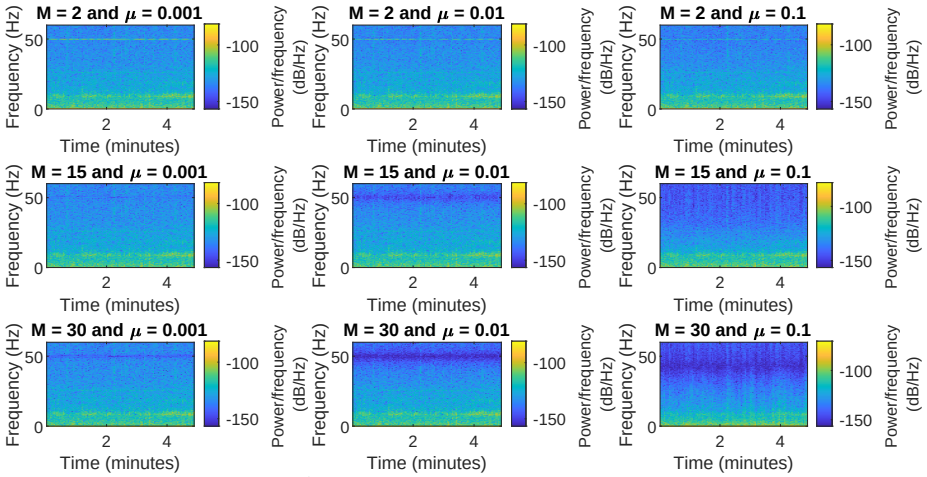


Figure 2.12: Plots of spectrogram of the ANC denoised signal for different filter orders M and LMS step-size μ.

As shown in Figure 2.12, the filter order used determines the amount of denoising done to the EEG data, for the same fixed step-size μ. Low filter orders (M=2) are not successfully suppressing the noise at 50Hz mainly due to under-fitting of noise component. As filter order increases to a value of M=15, the noise component is largely suppressed in the spectrum with almost no effect on the other spectrum components. However, for larger filter orders (M=30) the ANC method not only suppresses the noise component but also neighbouring spectral components between 40-60Hz (over-fits noise), distorting the spectrogram. Furthermore, as shown in Figure 2.12, the step-size μ also affects the performance of ANC method in de-noising signal, since the update of LMS weights are dependent on μ. For very low $\mu < 0.0001$ values (not shown in the plot), the ANC has poor performance due to the very low convergence speed in the LMS filter, thus noise component is not successfully supressed. On the other hand, larger μ values of 0.01 and 0.1, prevent the LMS filter to converge to the optimal solution, thus resulting in distortion of the signal since not only noise component is suppressed but also other spectral components. However, for a step-size of $\mu = 0.001$, the ANC method can successfully de-noise the signal, without affecting frequencies close to 50Hz. For these reasons, the filter order **M=15** and step-size $\boldsymbol{\mu = 0.001}$ was chosen, which achieve satisfactory performance in suppressing noise at 50Hz while not affecting other spectral components.

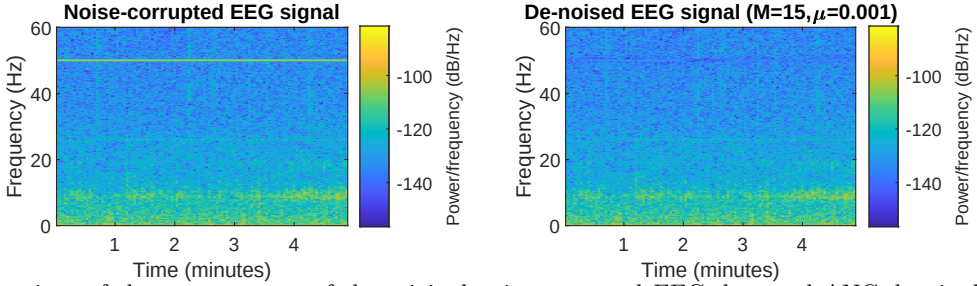


Figure 2.13: Comparison of the spectrograms of the original noise-corrupted EEG data and ANC denoised data using filter order M=15 and LMS step-size $\mu = 0.01$.

Using the parameters M=15 and $\mu = 0.001$ for ANC method, the spectrogram of original noise-corrupted EEG signal is

compared to that of the de-noised signal, shown in Figure 2.13, which illustrates that 50Hz components is successfully suppressed. To illustrate further the de-noising task of ANC, the periodogram of noise-corrupted and de-noised EEG data is plotted in Figure 2.14, using Averaged Periodogram approach with window size $\Delta t = 10s$. Figure 2.14 also shows the absolute error $|P_{original}(f) - P_{denoised}(f)|$ between noise-corrupted and de-noised data spectra, indicating that the maximum error occurs at frequency of 50Hz with magnitude around 30dB, and much lower magnitude (close to zero) for all other frequencies.

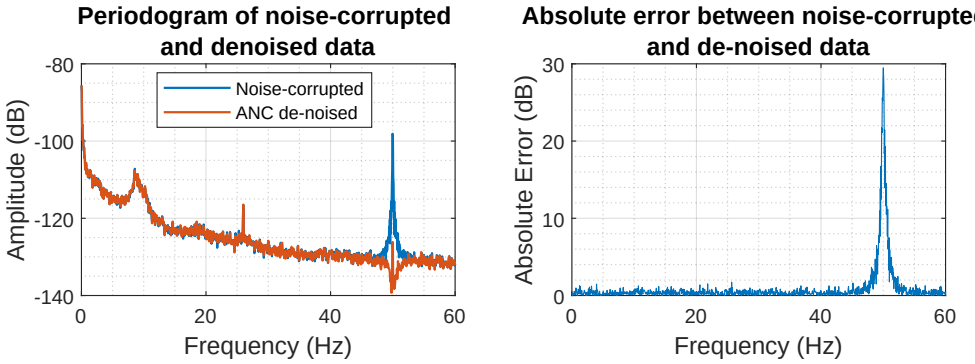


Figure 2.14: (Left) Comparison of spectrum for noise-corrupted and de-noised EEG data, using Averaged Periodogram approach with window size $\Delta t = 10s$. (Right) Absolute error between the noise-corrupted and de-noised EEG data, indicating maximum error at 50Hz where the 50Hz interference was removed by the ANC method.

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

Part a)

The Complex LMS (CLMS) and Augmented Complex LMS (ACLMS) algorithms were implemented in MATLAB for the identification of a first-order widely-linear-moving-average process (WLMA(1)) driven by circular white Gaussian noise $x(n)$, given by:

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

where $b_1 = 1.5 + j$ and $b_2 = 2.5 - 0.5j$. The real and imaginary parts of the WGN and WLMA(1) process are plotted in Figure 3.1 (circularity plots), which verifies that the WGN is indeed circular with circularity coefficient $|\rho| = \frac{|\mathbb{E}\{zz^*\}|}{\mathbb{E}\{zz^T\}} = 0.0012$ and shows that the WLMA(1) process is non-circular, as expected, with $|\rho| = 0.855$. It should be noted that circularity coefficient is the ratio of |pseudo-covariance| and covariance of a complex signal, with a lower value indicating a more circular signal.

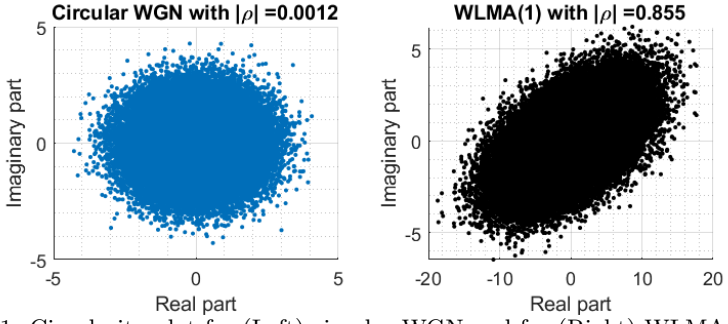


Figure 3.1: Circularity plot for (Left) circular WGN and for (Right) WLMA(1) process.

The learning curves, $10log|e(n)|^2$, for CLMS and ACLMS algorithm are shown in Figure 3.2, which are obtained by plotting an ensemble average of the learning curve of 100 independent realisations of process with 1000 data samples and using $\mu = 0.03$. The WLMA(1) process was identified using **Method 1** described in Section 2.2 a).

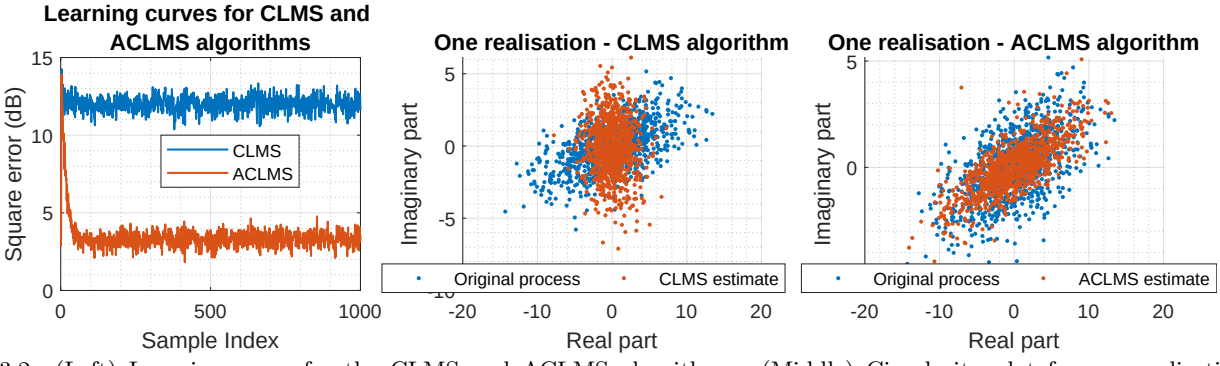


Figure 3.2: (Left) Learning curve for the CLMS and ACLMS algorithms. (Middle) Circularity plot for one realisation of WLMA(1) process and CLMS estimate. (Right) Circularity plot for one realisation of WLMA(1) process and ACLMS estimate.

In general, the CLMS approach is only valid for circular random variables, therefore it is expected that the CLMS algorithm will be unable to model successfully the WLMA(1) process. On the other hand, the ACLMS algorithm is valid for general complex data, both circular and non-circular, which means that it is expected to successfully learn the process parameters. This is verified by Figure 3.2, where ACLMS algorithm performs much better than CLMS approach, achieving a steady-state error of ≈ 3.4dB, compared to ≈ 12.6dB error of the CLMS algorithm. The difference is due to the fact that the ACLMS algorithm exploiting the full statistical information from the complex representation of data, which guarantees to capture the complete second-order statistics of in the data. This is achieved by using the additional weight vector $\mathbf{g}(n)$ and complex conjugate of input data $x^*(n)$. Therefore, Figure 3.2 verifies that ACLMS is more suitable for identifying non-circular processes. On the other hand, CLMS can successfully identity circular data only, since it considers only the covariance matrix of data and not the pseudo-covariance matrix for non-circular data (pseudo-covariance is zero for proper (circular) complex random variables, but has lower computational complexity than ACLMS algorithm).

Part b)

The bivariate wind data provided for three wind speed regimes (low,medium and high) in the East-West direction v_{east} and North-South direction v_{north} were combined into a complex valued signal of the form $v(n) = v_{east} + jv_{north}$. The circularity plots for the three speed regimes are shown in Figure 3.3. The circularity coefficient $|\rho|$ for each speed regime was also computed which quantifies the degree of non-circularity. Circularity coefficient varies from $0 \leq |\rho| \leq 1$, where $|\rho| = 0$ describes perfectly circular complex data, while $|\rho| = 1$ describes maximally non-circular complex data. For this data, $|\rho| = 0.1592$ for low-speed data, $|\rho| = 0.4543$ for medium-speed data and $|\rho| = 0.6237$ for high-speed data, therefore low-speed data is more circular compared to the other two regimes, with high-speed data being the least regular. This suggests that as wind speed, circularity of data decreases. Of course, circularity coefficient given here is only an estimate, as it is based on pseudo-covariance estimate which might not be zero even for perfectly circular data.

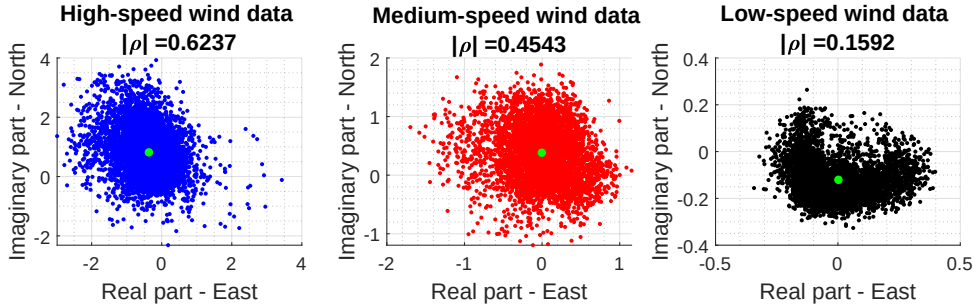


Figure 3.3: Circularity plots for the three wind speed regimes. Green dot denotes the center of data.

In general, complex-valued random variables are called circular if their probability distribution function (pdf) is not dependent on the angle, thus distribution is rotation invariant. In other words, if data is circular (rotation invariant), its pdf is only dependent of the Euclidean distance from the origin in the complex domain. Therefore, circularity coefficient quantifies how rotation invariant the distribution is about the origin of the complex domain. Despite the fact that the low-speed wind data has the least regular shape, the center of data is located closer to the origin, has the lowest difference between variance along the Real-axis and Imaginary axis and its |pseudo-covariance| is closer to zero compared to medium-speed and high-speed wind data.

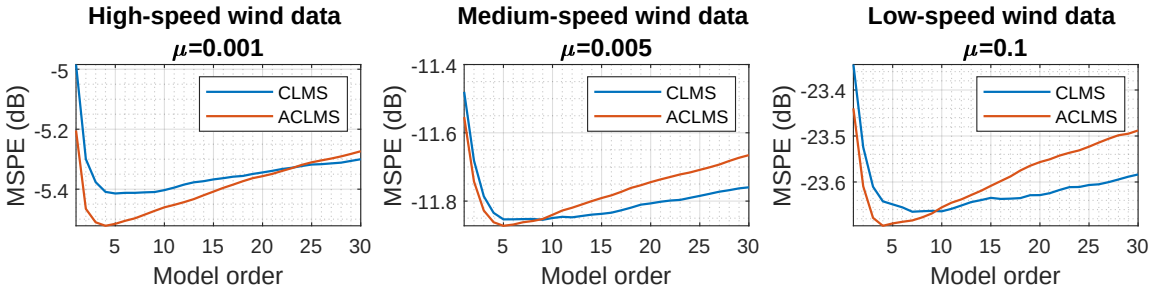


Figure 3.4: Plot of MPSE against model order for the three wind-data regimes, for CLMS and ACLMS algorithms.

The CLMS and ACLMS filters were used to perform one-step ahead prediction of the complex wind data from the three regimes. The filter length (model order) was varied for each algorithm, and performance of the two algorithms were quantified by Minimum Square Prediction Error (MSPE) measure, defined in 2.3 a). The learning rate μ was set to a different constant fixed value for each regime, determined experimentally to allow sufficient convergence of algorithms and low steady-state error. The chosen μ values were 0.1 for low-speed, 0.005 for medium-speed and 0.001 for high-speed data. The results are shown in Figure 3.4, were the performance (MSPE) of CLMS and ACLMS algorithms are compared for different filter lengths (model orders). By considering the minimum MSPE error achieved by the two algorithms, for CLMS and ACLMS, we can observe that the the widely linear ACLMS outperformed standard CLMS, for all three regimes due to the non-circular nature of the wind data. Specifically, for more non-circular data such us the high-speed wind data with large variation in their dynamics, the ACLMS algorithm significantly outperformed the CLMS algorithm. The increase in performance of prediction by ACLMS, however, decreases as the degree of

circularity within the signal increases. Using the minimum MSPE for ACLMS and CLMS, the absolute difference in performance was 1.989% for high-speed, 0.148% for medium-speed and 0.129% for low-speed wind data, thus verifying that ACLMS performs better for higher circularity coefficients (for increased non-circularity) and performance of ACLMS is comparable to CLMS for more circular data. In fact, the ACLMS outperforms CLMS for second order noncircular signals, and has identical performance for circular data, however due to higher complexity of ACLMS, CLMS can be used instead for more circular data ($|\rho|$ close to 0). It should be noted that the minimum MSPE value achieved by the two algorithms in the three regimes is not the same due to the different amplitude of wind-data in the three data sets. To mitigate this, we could instead standardise first the data before comparisons, which was not done in this case explaining why percentage differences were used.
For low model orders, both algorithms have an increased error due to under-fitting of the data, however the ACLMS has better performance than CLMS due to the extra parameters taken into consideration, thus compensating more effectively the under-fitting of the data. This occurs until both algorithms reach their minimum value (order at which minimum occurs provided in Table 10). On the other hand, as model order increases above the value where minimum MSPE occurs, the one-step ahead prediction performance of both algorithms starts to degrade due to over-fitting occurring. Due to the higher number of parameters involved in the ACLMS algorithm, over-fitting effects become more pronounced and after a certain model order the performance of CMLS is better than ACLMS. As the circularity of data increases, we can observe that the model order at which MSPE of CLMS algorithm becomes lower than that of ACLMS algorithm, decreases. For example, $MSPE_{CLMS} > MSPE_{ACLMS}$ when order$>$ 23 for high-speed and oder$>$ 9 for low-speed data. Additionally, due to more parameters controlled by ACLMS, model order at which minimim MSPE occurs is always lower (high and medium speed data) or equal (low speed data) to that of CLMS. Finally, due to the higher number of parameters tuned by ACLMS algorithm, the rate of increase of MSPE due to overfitting is higher for ACLMS compared to CLMS as model order increases (higher slope of ACLMS error compared to CLMS after minimum MSPE occurs).

Table 10: Model orders at which $MSPE_{min}$ occurs, with the corresponding MSPE values for CLMS and ACLMS.

	High wind		**Medium wind**		**Low wind**	
Algorithm	**CMLS**	**ACMLS**	**CMLS**	**ACMLS**	**CMLS**	**ACMLS**
MSPE$_{min}$ (dB)	-5.414	-5.522	-11.855	-11.872	-23.665	-23.695
Model order at MSPE$_{min}$	5	4	9	5	7	7

Part c)
For this part, different three-phase voltages were generated with various magnitude and phase values, to produce balanced and unbalanced power systems. For each case, the Clarke transform was used to project the three-phase system onto two orthogonal axes to form the the **I** and **Q** components $v_\alpha(n)$ and $v_\beta(n)$, which were then represented as a complex Clarke $(\alpha - \beta)$ voltage $v(n) = v_\alpha(n) + jv_\beta(n)$. The balanced system was generated by using the same peak voltage for each signal, i.e. $V_a(n) = V_b(n) = V_c(n) = V$, and same phase shift of exactly 120 degrees, i.e. $\Delta_b = \Delta_c = 0$. On the other hand, the unbalanced systems were generated by changing the magnitude and/or phase of one or more signals. Figures 3.5 and 3.6, show the circularity plots for each system (of the complex $\alpha - \beta$ voltages).

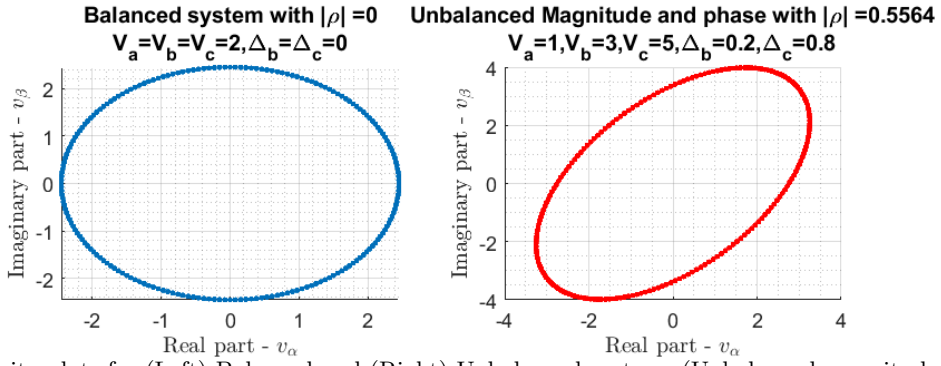


Figure 3.5: Circularity plots for (Left) Balanced and (Right) Unbalanced systems (Unbalanced magnitude and phase).

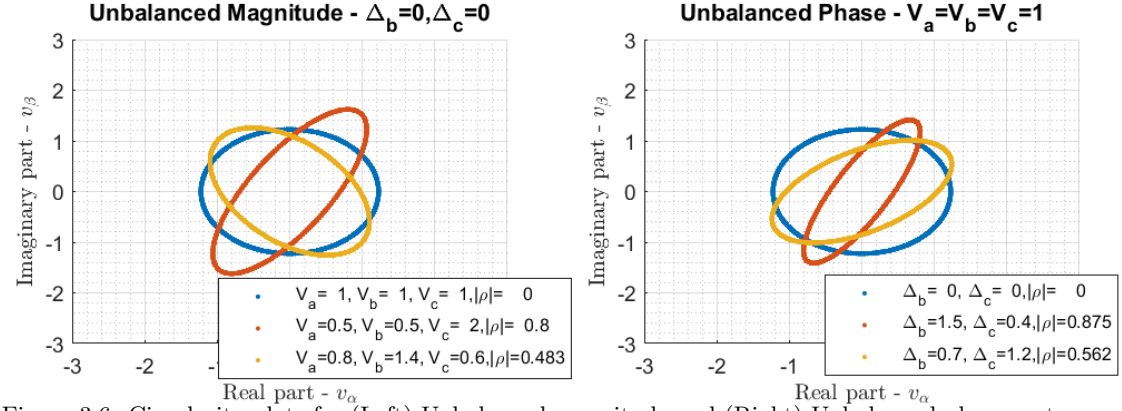


Figure 3.6: Circularity plots for (Left) Unbalanced magnitude and (Right) Unbalanced phase systems.

As Figures 3.5 and 3.6 suggest, we can identify if the system is balanced or unbalanced by considering the shape of the circularity diagrams. A balanced system results in a circular trajectory in the circularity diagram with perfect symmetry about the origin of the plot. On the other hand, unbalanced systems have elliptical trajectories in the circularity diagrams. Additionally, the circularity coefficient for a perfectly balanced system is equal to 0, while for an unbalanced system varies between $0 < |\rho| \leq 1$. Therefore, we can use the circularity diagram to detect if there is a fault (unbalance) in the system when the trajectory is not circular, and we can quantify the degree of severity of the fault using the circularity coefficient. A higher deviation of circularity coefficient from zero, suggests a more severe fault in system.

Part d)

Using the strictly linear and widely linear autoregressive models of order 1, given by Equations 3.2 and 3.3, we can compute the frequency of the balanced and unbalanced complex Clarke $(\alpha - \beta)$ voltages $v(n)$.

$$\text{Strictly Linear}: \ v(n+1) = h^*(n)v(n) \tag{3.2}$$

$$\text{Widely Linear}: \ v(n+1) = h^*v(n) + g^*(n)v^*(n) \tag{3.3}$$

For the balanced system, Clarke voltage $v(n)$ is given by:

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \tag{3.4}$$

where f_s is the sampling frequency, ϕ is the phase shift and n is the time index. By substituting Equation 3.4 into the strictly linear AR(1) model in Equation 3.2 we obtain:

$$\begin{aligned}
\sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} &= h^*(n) \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \\
e^{j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} &= h^*(n) e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \\
e^{j(2\pi \frac{f_0}{f_s} n + \phi)} e^{j(2\pi \frac{f_0}{f_s})} &= h^*(n) e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \\
e^{j(2\pi \frac{f_0}{f_s})} &= h^*(n)
\end{aligned} \tag{3.5}$$

We can then define $h(n) = |h(n)|e^{j\phi_h}$ with its conjugate $h^*(n) = |h(n)|e^{-j\phi_h}$, or equivalently $h(n) = |h(n)|e^{-j\phi_h}$ with $h^*(n) = |h(n)|e^{j\phi_h}$. Since the negative sign in the phase of h(n) only indicates the direction of rotation of signal, we can choose any of the two definitions for h(n). For this derivation the second definition is used so that we arrive at a positive expression for frequency f_0. We can also express $h^*(n)$, in terms of of real and imaginary components of h(n) such that:

$$h^*(n) = e^{jarctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right)} \tag{3.6}$$

Substituting Equation 3.6 in Equation 3.5, we obtain:

$$e^{j(2\pi \frac{f_0}{f_s})} = |h(n)| e^{jarctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right)} \tag{3.7}$$

Since two complex values are equal only if their magnitudes are equal, then we know that $|h(n)| = 1$ since magnitude of exponential is 1, and also that:

$$\begin{aligned}
\frac{2\pi f_0(n)}{f_s} &= \arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \\
f_0(n) &= \frac{f_s}{2\pi} \arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right)
\end{aligned} \tag{3.8}$$

which gives the desired result. It should be noted that f_0 is a function of n since right-hand side of Equation 3.8 is also a function of n and all other terms are constants.

In the case of an unbalanced system, the general form of the complex-valued voltage that arises from the Clarke Transform is given by:

$$v(n) = A(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)} + B(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} \tag{3.9}$$

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

where $V_a(n)$,$V_b(n)$ and $V_c(n)$ are the peak values of the three-phase voltages of the power system. Substituting Equation 3.9 in left-hand side and right-hand side of widely linear AR(1) model definition (Equation 3.3), we obtain the following two equations:

- $v(n+1) = A(n+1)e^{j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} + B(n+1)e^{-j(2\pi \frac{f_0}{f_s}(n+1) + \phi)}$ (3.10)
- $v(n+1) = h^*(n)A(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)} + h^*(n)B(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} + g^*(n)A^*(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} + g^*(n)B^*(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)}$ (3.11)

Equating Equations 3.10 and 3.11:

$$\begin{aligned}
A(n+1)e^{j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} + B(n+1)e^{-j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} &= h^*(n)A(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)} + h^*(n)B(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} \\
&\quad + g^*(n)A^*(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} + g^*(n)B^*(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \\
A(n+1)e^{j(2\pi \frac{f_0}{f_s} n + \phi)} e^{j(2\pi \frac{f_0}{f_s})} + B(n+1)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} e^{-j(2\pi \frac{f_0}{f_s})} &= h^*(n)A(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)} + h^*(n)B(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} \\
&\quad + g^*(n)A^*(n)e^{-j(2\pi \frac{f_0}{f_s} n + \phi)} + g^*(n)B^*(n)e^{j(2\pi \frac{f_0}{f_s} n + \phi)}
\end{aligned} \tag{3.12}$$

By factorising Equation 3.12, and collecting common exponential terms, i.e. separating terms associated with $e^{j(2\pi\frac{f_0}{f_s}n+\phi)}$ and terms associated with $e^{-j(2\pi\frac{f_0}{f_s}n+\phi)}$, we can obtain the following relations:

$$\bullet\ A(n+1)e^{j(2\pi\frac{f_0}{f_s})} = h^*(n)A(n) + g^*(n)B^*(n) \tag{3.13}$$

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

If we assume that the three-phase voltage magnitudes $V_a(n)$,$V_b(n)$ and $V_c(n)$ in Equation 3.9(also involved in expressions for $A(n)$ and $B(n)$) are constant with time or their rate of change with time is negligible, then we can set $A(n+1) \approx A(n)$ and $B(n+1) \approx B(n)$, such that:

$$\bullet\ A(n)e^{j(2\pi\frac{f_0}{f_s})} = h^*(n)A(n) + g^*(n)B^*(n) \quad \Rightarrow \quad e^{j(2\pi\frac{f_0}{f_s})} = h^*(n) + g^*(n)\left(\frac{B^*(n)}{A(n)}\right) \tag{3.15}$$

$$\bullet\ B(n)e^{-j(2\pi\frac{f_0}{f_s})} = h^*(n)B(n) + g^*(n)A^*(n) \quad \Rightarrow \quad e^{-j(2\pi\frac{f_0}{f_s})} = h^*(n) + g^*(n)\left(\frac{A^*(n)}{B(n)}\right) \tag{3.16}$$

Taking the conjugate of Equation 3.16 and equating this to Equation 3.15:

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

By letting $K(n) = B^*(n)/A(n)$, and substituting this into Equation 3.17, we can obtain a relation between $K(n)$, $h(n)$ and $g(n)$:

$$\begin{aligned} h^*(n) + g^*(n)K(n) &= h(n) + g(n)\left(\frac{1}{K(n)}\right) \\ g^*(n)K^2(n) + [h^*(n) - h(n)]K(n) - g(n) &= 0 \end{aligned} \tag{3.18}$$

We can then solve for $K(n)$ to express it in terms of $h(n)$ and $g(n)$:

$$K(n) = \frac{(h(n) - h^*(n)) \pm \sqrt{(h^*(n) - h(n))^2 + 4g^*(n)g(n)}}{2g^*(n)} \tag{3.19}$$

Since we defined $h(n) = |h(n)|e^{-j\phi_h}$ and $h^*(n) = |h(n)|e^{j\phi_h}$, then we know that $h(n) - h^*(n) = -2j\Im\{h(n)\}$ and $(h^*(n) - h(n)) = -4\Im^2\{h(n)\}$. Also, $g^*(n)g(n) = |g(n)|^2$. Therefore we can simplify Equation 3.19 by:

$$\begin{aligned} K(n) &= \frac{-2j\Im\{(h(n)\} \pm \sqrt{-4\Im^2\{h(n)\} + 4|g(n)|^2}}{2g^*(n)} \\ K(n) &= j\frac{-\Im\{(h(n)\} \pm \sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{g^*(n)} \end{aligned} \tag{3.20}$$

Substituting expression for $K(n)$ (Equation 3.20) into Equation 3.17, we obtain:

$$\begin{aligned} e^{j(2\pi\frac{f_0}{f_s})} &= h^*(n) + jg^*(n)\left(\frac{-\Im\{(h(n)\} \pm \sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{g^*(n)}\right) \\ &= h^*(n) - j\Im\{h(n)\} \pm j\sqrt{\Im^2\{h(n)\} - |g(n)|^2} = \Re\{h(n)\} \pm j\sqrt{\Im^2\{h(n)\} - |g(n)|^2} \end{aligned} \tag{3.21}$$

where we used the fact that $h^*(n) - j\Im\{h(n)\} = \Re\{h(n)\}$. We can express the right-hand side of Equation 3.21 in exponential form, given by:

$$e^{j(2\pi\frac{f_0}{f_s})} = \Re\{h(n)\} \pm j\sqrt{\Im^2\{h(n)\} - |g(n)|^2} = Ce^{jarctan\left(\frac{\pm j\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right)} \tag{3.22}$$

where C is a real constant, with value of 1 in order for equality to hold. Since the sign of phase only indicated direction of rotation, we can choose positive sign in the argument of arctan. By equating phases of both terms in Equation 3.22:

$$\frac{2\pi f_0(n)}{f_s} = \arctan\left(\frac{\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right) \quad \Rightarrow \quad f_0(n) = \frac{f_s}{2\pi}\arctan\left(\frac{\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right) \tag{3.23}$$

which gives the desired result. Since the right-hand side of Equation 3.23 is a function of n, the f_0 is also a function of n.

Part e)

The CLMS and ACLMS algorithms were implemented on the Clarke ($\alpha - \beta$) voltages for a balanced system (with $|\rho| = 0$) and an unbalanced system (unbalanced phase and magnitude system with $|\rho| = 0.648$). Using Equations 3.8 and 3.23, an estimate for the nominal frequency f_0 of balanced and unbalanced system voltages were obtained. For this task nominal frequency for both systems was set to 50Hz (resembling UK power line frequency), data size N=5000, sampling frequency f_s=1kHz, learning rate $\mu = 0.05$ and phase shift of three-phase voltages ϕ=0. The parameter values used for the generation of balanced and unbalanced systems are provided in Table 11.

Table 11: Parameters used for the generation of balanced and unbalanced system voltages.

	$\mathbf{V_a}$	$\mathbf{V_b}$	$\mathbf{V_c}$	$\mathbf{\Delta_b}$	$\mathbf{\Delta_c}$
Balanced System	1	1	1	0	0
Unbalanced System	1	0.8	1.2	0.4	0.9

It should be noted, that since arguments of arctan in Equations 3.8 and 3.23 involve division by $\Re\{h(n)\}$, the initial value $h(0)$ was set to 1, to avoid presence of `NaN` values in the $\hat{f}_0$ estimations (prevents division of zero by zero). This also prevents initial overshoot in $\hat{f}_0$ by ACLMS algorithm.

Figure 3.7 shows the time-evolution of frequency f_0 estimation for the two systems using CLMS and ACLMS algorithms.

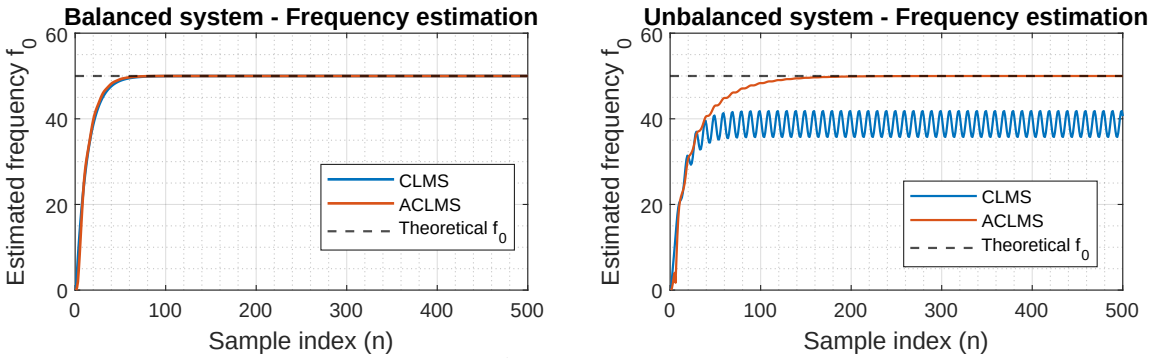


Figure 3.7: Frequency f_0 estimations using CLMS and ACLMS algorithms. To illustrate clearly the performances of the two algorithms, only 500 sample points are shown.

The general form of complex valued voltage, due to Clarke Transform, is:

$$v(n) = A(n)e^{j(2\pi\frac{f_0}{f_s}n+\phi)} + B(n)e^{-j(2\pi\frac{f_0}{f_s}n+\phi)} \tag{3.24}$$

where $B(n) = 0$ for balanced systems and $B(n) \neq 0$ for unbalanced systems.

For the case of balanced system, both CLMS and ACLMS algorithms converge to the true value of frequency at 50Hz, with zero bias, after approximately 60 time-steps (approximately same rate of convergence). This shows that ACLMS has identical performance to CLMS for circular data. This is because the pseudo-covariance of data is essentially zero. therefore both algorithms have the ability to completely describe the second-order statistical behavior of data. As observed from simulations, the real and imaginary components of the extra parameter $g(n)$ in ACLMS algorithm are both converging to a value of zero, thus for circular data Equation 3.3 is reduced to Equation 3.2. In other words, the widely-linear model for circular data is the same as strictly-linear model when convergence is reached. Finally, due to convergence of $g(n)$ to zero, the equation for frequency equation (Equation 3.23) of ACLMS is the same as that of CLMS (Equation 3.8) at convergence.

For the case of unbalanced system, ACLMS converges to the true frequency value at 50Hz after approximately 200 iterations, while CLMS oscillates around an incorrect frequency value after 100 iterations, giving an average estimate of $\hat{f}_0 = 38.7$Hz, which is not the correct frequency estimate (biased). This shows that ACLMS has the ability to correctly identify correct frequency estimate for both circular and non-circular data, while CLMS gives correct frequency estimate only for circular data. The pseudo-covariance of non-circular data, i.e. for the unbalanced system, is not zero and therefore CLMS cannot fully utilise all the available information within the data to describe completely the second-order statistics of the system. As a result, ACLMS outperforms CLMS for second order noncircular signals. Additionally, as observed from simulations, the real and imaginary components of the extra parameter $g(n)$ in ACLMS algorithm do not converge to zero, meaning that widely-linear and strictly-linear models are not equivalent. In fact, the strictly linear model yields biased estimates when the system is unbalanced, since the conjugate component of signal in Equation 3.24 is not captured by the model (equivalent to description of an elliptical trajectory by a circular trajectory). The parameter $g(n)$ in ACLMS gives an additional degree of freedom in the widely-linear model, which significantly improves the performance of the algorithm. The fact that convergence speed of ACLMS is lower for non-circular data compared to circular data is due to the fact that $g(n)$ is not converging to zero. Additionally, it was also observed that the lower the circularity coefficient of data (less 'unbalanced' data), the smaller the deviation of the frequency estimate of CLMS algorithm. For example, if $|\rho| = 0.272$ then average frequency (around which CLMS estimate oscillates) is 48.3Hz.

To conclude, simulations verify that ACLMS algorithm is able to estimate the frequency for both circular (balanced) and non-circular (unbalanced) voltages, while CLMS can only estimate frequency for circular (balanced) voltages only.

3.2 Adaptive AR Model Based Time-Frequency Estimation

Part a)

For this section, a frequency modulated (FM) signal (length 1500) was generated given by $y(n) = e^{j(\frac{2\pi}{f_s}\phi(n))} + \eta(n)$, where $\eta(n)$ is circular complex white Gaussian noise with zero mean and variance 0.05, and sampling frequency $f_s = 1500Hz$. The phase term $\phi(n)$ is equal to $\phi(n) = \int f(n)dn$, where f(n) is time-dependent (non-stationary) frequency of the complex signal. Figure 3.8 shows how $\phi(n)$ and $f(n)$ vary with time. As shown in Figure 3.8, time-variation of frequency $f(n)$ is composed of a constant region, a linear region and a quadratic region.

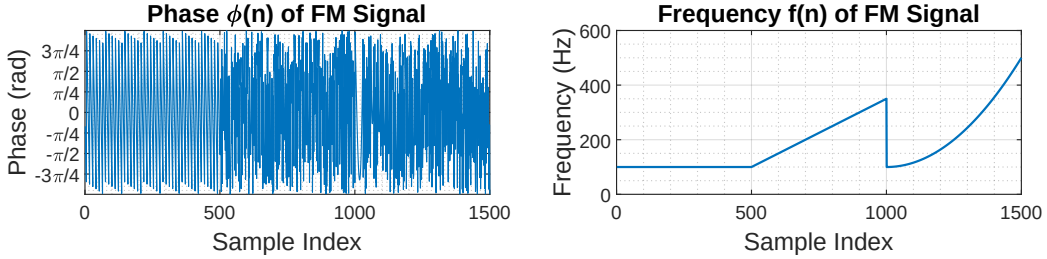


Figure 3.8: Time-variation of (Left) phase $\phi(n)$ and of (Right) frequency $f(n)$ of FM signal $y(n)$.

The complete signal was then modelled as an AR(1) process and the AR coefficient was obtained using `aryule` function

in MATLAB. The power spectrum estimate of the signal was then obtained using `freqz` function and is shown in Figure 3.9.

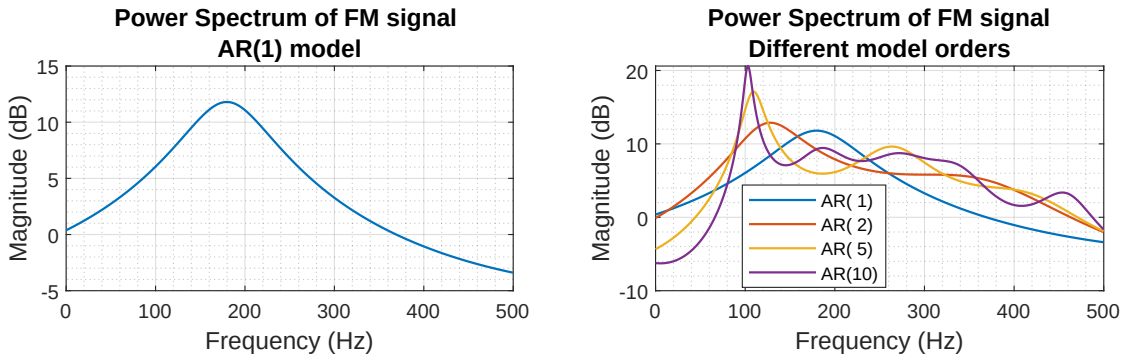


Figure 3.9: Power spectrum estimate when signal is modelled by (Left) an AR(1) model and (Right) AR models of different orders.

Since the `aryule` function assumes that the signal is stationary, the power spectrum estimate (Figure 3.9) only shows a single spectral peak (at $\approx 177.7Hz$), which represents the average frequency of the signal over time, thus the model cannot adapt to the time-varying $f(n)$. It is evident from Figure 3.9, that even a higher order model of the complete signal cannot capture the changes in frequency. For higher model orders, however, the number of spectral peaks increases and specifically for model order of 10, we can identify a peak at 100Hz, corresponding to the constant region of $f(n)$. On the other hand, the remaining frequency variations in the signal cannot be clearly or correctly identified since they occur in the signal a maximum of two times. Therefore the AR models of the complete signal cannot estimate the frequency changes correctly.

Since the time-variation of frequency $f(n)$ is composed of three regions, we can also model each one as an AR process and obtain the power spectrum estimate for each case, thus increasing resolution of estimates by discarding discontinuities in $f(n)$. The results are shown in Figure 3.10.

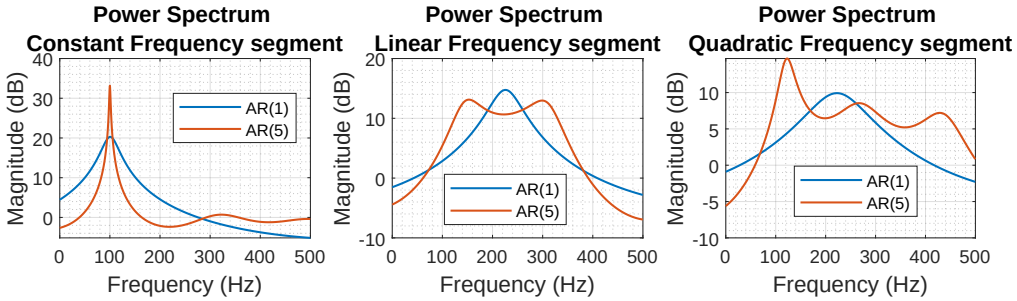


Figure 3.10: Power spectrum estimate when signal is modelled by AR models of different orders for the three segments (500 sample long) of $f(n)$ variation, i.e. (Left) constant region, (Middle) linear region and (Right) quadratic region.

As Figure 3.10 shows, the AR(1) model can accurately capture the constant frequency in the first region. Although more information is provided in the spectra of linear and quadratic regions (different peaks giving average frequency within the segment), due to non-stationarity of the data in linear and quadradic region, the AR(1) model cannot capture the changes in frequency. If AR model order is increases, the number of spectral peaks increases providing more information about the range of frequencies present in each segment (due to flat regions in spectrum with high amplitude). However, as we saw before for the complete signal, increasing model order does not improve significantly the results.

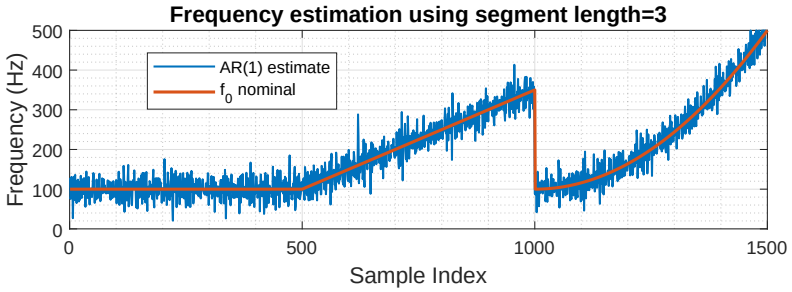


Figure 3.11: Frequency estimate for each time sample for AR(1) model, using 3-sample long segments with 60% overlap.

To increase further the resolution of the Yule-Walker spectrum estimation, we could potentially decrease the segment size to a very low value. Figure 3.11 was obtained by using 3-sample long data segments with 66% overlap, where for each case the AR(1) coefficient was obtained. For each segment, the power spectrum was computed from which frequency with maximum amplitude was extracted to form this plot. In this case, the AR(1) model assumed stationarity of each segment as an approximation, however, as results show it is able to capture the changes in frequencies. Increasing resolution (decrease in bias) by decreasing segment size, comes in the expense of higher variance as found from simulations. For example, when segment size=5, variance of $\hat{f}_0$ decreases but bias increases (higher deviation from true value).

Part b)

The CLMS algorithm was implemented on the FM signal $y(n)$ to estimate the AR coefficient and also the time-frequency spectrum using the code provided in the coursework instructions. Since the data is highly circular ($|\rho| = 0.018$), the CLMS algorithm can model the process as $y(n) = a_1^* y(n)$ and has the ability, by iteratively and adaptively updating coefficient estimate $\hat{a}_1^*$, to capture the non-stationary frequency changes in the data. The iterative update of coefficient considers only the past samples of signal $y(n)$ and not the entire data at once as is done using `aryule` (block-estimate of AR coefficient for entire signal), therefore CLMS can adapt coefficient according to the changes occurring in signal. The time-frequency spectrum estimates for CLMS method using different learning rates μ is shown in Figure 3.12. A good choice of $\mu = 0.01$, since it can track sufficiently the changes in frequency of signal and balance the trade-off between converge speed (convergence to true frequency) and steady-state variance. For low μ values ($\mu \leq 0.005$) convergence of CLMS is very slow, which means that AR coefficient is not updating fast enough to capture changes in linear and quadratic variations in $f(n)$ and spectral estimates do not reach the true values. On the other hand, large μ values ($\mu \geq 0.5$) introduce noise in the spectrum causing frequency estimates to oscillate around the true value (amplitude of oscillations increases with μ). It was also empirically observed that if higher order of AR model is used, the width of the spectral peak at each time instance decreases thus increasing precision of estimate, however if order is increased above a certain value false peaks can be detected in spectrum (over-fitting).

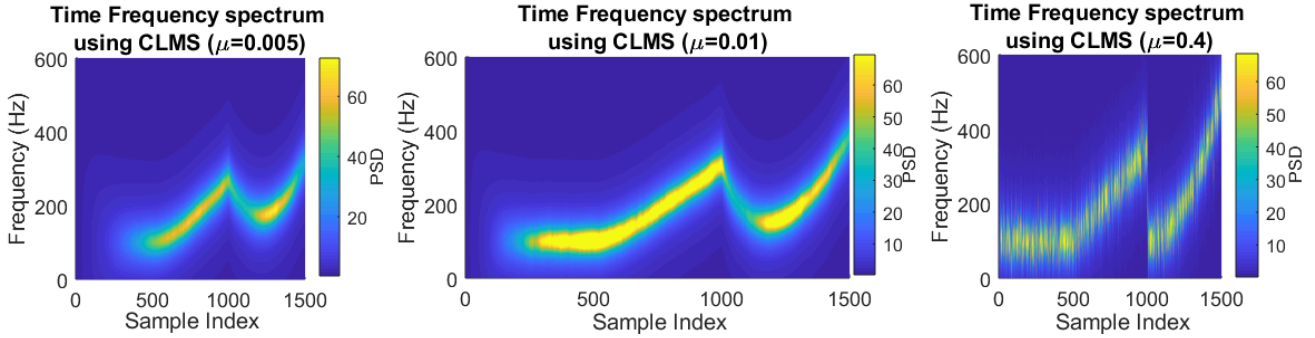


Figure 3.12: Time-frequency spectrum by CLMS algorithm for AR (1) estimates using different learning rates μ. PSD is the abbreviation to Power Spectral Density.

3.3 A Real Time Spectrum Analyser Using Least Mean Square

Part a)

A signal $y(n)$ can be estimated by a linear combination of N harmonically related sinusoids, to yield $\hat{y}(n)$ given by:

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k) e^{j2\pi kn/N} \tag{3.25}$$

where $w(k)$ are the unknown weights. This can also be expressed in vector form as:

$$\begin{bmatrix} \hat{y}(0) \\ \hat{y}(1) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{j\frac{2\pi}{N}(1)(1)} & \dots & e^{j\frac{2\pi}{N}(1)(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{j\frac{2\pi}{N}(N-1)(1)} & \dots & e^{j\frac{2\pi}{N}(N-11)(N-1)} \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ \vdots \\ w(N-1) \end{bmatrix} \quad \Rightarrow \quad \hat{\mathbf{y}} = \mathbf{F}\mathbf{w} \tag{3.26}$$

The vector $\mathbf{w}$, from a least squares (LS) perspective, can be found by minimising the cost function $J(\mathbf{w})$, which is the sum of squared errors between estimated signal $\hat{y}(n)$ and true signal $y(n)$. The cost function in vector form is given by:

$$J(\mathbf{w}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \|\mathbf{y} - \mathbf{F}\mathbf{w}\|^2 = (\mathbf{y} - \mathbf{F}\mathbf{w})^H(\mathbf{y} - \mathbf{F}\mathbf{w}) \tag{3.27}$$

where $\mathbf{y}$ is the vector containing the actual data $\mathbf{y} = [y(0), y(1), ..., y(N-1)]^T$. We can find the LS solution, i.e. find optimal w such that $\mathbf{w}_{opt} = \min_{\mathbf{w}} J(\mathbf{w})$. To find the LS solution, we start by first expanding Equation 3.27:

$$\begin{aligned} J(\mathbf{w}) &= (\mathbf{y} - \mathbf{F}\mathbf{w})^H(\mathbf{y} - \mathbf{F}\mathbf{w}) = (\mathbf{y}^H - \mathbf{w}^H\mathbf{F}^H)(\mathbf{y} - \mathbf{F}\mathbf{w}) \\ &= \mathbf{y}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}\mathbf{w} - \mathbf{w}^H\mathbf{F}^H\mathbf{y} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w} \end{aligned} \tag{3.28}$$

To find $\mathbf{w}_{opt}$ that minimises the cost function, we can utilise the fact that at $\mathbf{w} = \mathbf{w}_{opt}$ the derivative of the cost function J with respect to the weights $\mathbf{w}$ is zero. Therefore, taking the derivative of Equation 3.28 with respect to $\mathbf{w}$, we obtain:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 0 - \mathbf{F}^H\mathbf{y} - \mathbf{F}^H\mathbf{y} + 2\mathbf{F}^H\mathbf{F}\mathbf{w} = 2\mathbf{F}^H\mathbf{F}\mathbf{w} - 2\mathbf{F}^H\mathbf{y} \tag{3.29}$$

Setting Equation 3.29 equal to zero, obtain the LS solution given by:

$$\begin{aligned} \mathbf{F}^H\mathbf{F}\mathbf{w}_{opt} &= \mathbf{F}^H\mathbf{y} \\ \mathbf{w}_{opt} &= (\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{y} \end{aligned} \tag{3.30}$$

Matrix $\mathbf{F}$ is full-rank, since all its column vectors are orthogonal. As a result, matrix $\mathbf{F}^H\mathbf{F}$ is also full-rank. In fact, matrix $\mathbf{F}^H\mathbf{F} = N\mathbf{I}$, which means its a symmetric diagonal matrix with positive diagonal elements, ensuring that its determinant is $\neq 0$ and thus it is invertible. As we know, we can approximate a signal $x(n)$ using the Discrete Fourier transform (DFT) and Inverse Discrete Fourier transform (IDFT), to obtain $\hat{x}(n)$:

$$\underline{\text{DFT:}}\ X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \quad \text{and} \quad \underline{\text{IDFT:}}\ \hat{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N} \tag{3.31}$$

where $X(k)$ are the Fourier coefficients. We can express the IDFT formula in Equation 3.31 also in vector form as:

$$\begin{bmatrix} \hat{x}(0) \\ \hat{x}(1) \\ \vdots \\ \hat{x}(N-1) \end{bmatrix} = \frac{1}{N} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{j\frac{2\pi}{N}(1)(1)} & \dots & e^{j\frac{2\pi}{N}(1)(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{j\frac{2\pi}{N}(N-1)(1)} & \dots & e^{j\frac{2\pi}{N}(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{bmatrix} \quad \Rightarrow \quad \hat{\mathbf{x}} = \mathbf{F}_{DFT}\mathbf{X} \tag{3.32}$$

By comparing Equation 3.32 with Equation 3.26, we can identify that $N\mathbf{F}_{DFT} = \mathbf{F}$. This shows that the estimate of signal $y(n)$ given by equation 3.26, is equivalent to the IDFT of signal, where the matrix $\mathbf{F}$ is scaled by a factor of $(1/N)$ to produce $\mathbf{F}_{DFT}$. Using IDFT, the column vectors of matrix $\mathbf{F}_{DFT}$ are orthonormal, while for $\mathbf{F}$ are orthogonal, due to $(1/N)$ factor in $\mathbf{F}_{DFT}$. Additionally, by expressing the IDFT as in Equation 3.32 as an LS minimisation problem, we can obtain Fourier Coefficients $\mathbf{X}$ using an expression equivalent to Equation 3.30, and thus obtain a relationship between LS solutions $\mathbf{w}_{opt}$ and $\mathbf{X}_{opt}$ for same data $y(n)$:

$$\mathbf{X}_{opt} = (\mathbf{F}_{DFT}^H\mathbf{F}_{DFT})^{-1}\mathbf{F}_{DFT}^H\mathbf{y} = (\frac{1}{N^2}\mathbf{F}^H\mathbf{F})^{-1}\frac{1}{N}\mathbf{F}^H\mathbf{y} = N(\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{y} = N\mathbf{w}_{opt} \tag{3.33}$$

We know that $\mathbf{F}^H\mathbf{F} = (N)\mathbf{I}$ and $\mathbf{F}_{DFT}^H\mathbf{F}_{DFT} = (1/N)\mathbf{I}$ and we can use this idea to show that the first equality of Equation 3.33 is therefore equivalent to the DFT of the signal $y(n)$ when the LS error is minimised (optimal solution). Utilising also the orthogonality property of matrices $\mathbf{F}_{DFT}$ and $\mathbf{F}$, i.e. using $\mathbf{F}_{DFT}^H = N\mathbf{F}_{DFT}^{-1}$ and $\mathbf{F}^H = N\mathbf{F}^{-1}$:

- $\mathbf{X}_{opt} = (\mathbf{F}_{DFT}^H\mathbf{F}_{DFT})^{-1}\mathbf{F}_{DFT}^H\mathbf{y} = N\mathbf{I}\mathbf{F}_{DFT}^H\mathbf{y} = N\mathbf{F}_{DFT}^H\mathbf{y}$ (3.34)

$$= \mathbf{F}_{DFT}^{-1}\mathbf{y} \quad = \quad \underline{\text{DFT}} \quad \Rightarrow \quad \hat{\mathbf{y}} = \mathbf{F}_{DFT}\mathbf{X}_{opt} \quad = \quad \underline{\text{IDFT}} \tag{3.35}$$

- $\mathbf{w}_{opt} = (\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{y} = \frac{1}{N}\mathbf{I}\mathbf{F}^H\mathbf{y} = \frac{1}{N}\mathbf{F}^H\mathbf{y}$ (3.36)

$$= \mathbf{F}^{-1}\mathbf{y} \quad = \quad \underline{\text{DFT-equivalent}} \quad \Rightarrow \quad \hat{\mathbf{y}} = \mathbf{F}\mathbf{W}_{opt} \quad = \quad \underline{\text{IDFT-equivalent}} \tag{3.37}$$

Finally, by substituting the relation $N\mathbf{F}_{DFT} = \mathbf{F}$ in Equation 3.34, we can identify that we can obtain the DFT coefficients by $\mathbf{X}_{opt} = \mathbf{F}^H\mathbf{y} = N\mathbf{F}^{-1}\mathbf{y}$. Therefore estimate of $\mathbf{y}$ is given by $\hat{\mathbf{y}} = (1/N)\mathbf{F}\mathbf{X}_{opt}$, agreeing with definition of DFT in Equation 3.31.

Part b)

The DFT matrix $\mathbf{F}_{DFT}$ is an orthogonal matrix since its column vectors are mutually orthogonal. In other words, if we denote each of the N column vectors in matrix $\mathbf{F}_{DFT}$ as $\mathbf{f}_k = \frac{1}{N}[1, e^{j2\pi k/N}, \dots, e^{j2\pi k(N-1)/N}]^T$, the Hermitian inner product between the column vectors is given by:

$$< \mathbf{f}_k, \mathbf{f}_m >= \mathbf{f}_k^H \cdot \mathbf{f}_m = \sum_{k=0}^{N-1} \frac{1}{N^2} e^{j\frac{2\pi n(m-k)}{N}} = \begin{cases} \frac{1}{N}, & \text{if } k = m \\ 0, & \text{if } k \neq m \end{cases} \tag{3.38}$$

Additionally, each column vector due to $(1/N)$ scale factor has magnitude of 1, so it is a unit vector and therefore matrix $\mathbf{F}_{DFT}$ is orthonormal since the columns vectors form an orthonormal set, of basis functions. The estimate of data $\hat{\mathbf{y}} = \mathbf{F}_{DFT}X$, where effectively the data $y(n)$ is decomposed into a sum of orthogonal basis functions. The Fourier coefficients are obtained by $\mathbf{X} = \mathbf{F}_{DFT}^{-1}\mathbf{y} = N\mathbf{F}_{DFT}^H\mathbf{y}$, where the matrix $\mathbf{F}_{DFT}^H$ is also an orthonormal matrix. Therefore, DFT effectively projects data $\mathbf{y}$ onto an N-dimensional orthogonal subspace (Fourier domain) spanned by the orthonormal set of basis vectors, which are the column vectors of DFT matrix $\mathbf{F}_{DFT}$. This projection is done through an inner product between data $\mathbf{y}$ and each of the basis vectors, which are the column vectors of the complex conjugate of $\mathbf{F}_{DFT}$. Therefore, this matrix is considered as a change of basis matrix, since it allows description of signal $\mathbf{y}$ in signal vector space (Time-domain) by complex exponential basis set (Frequency/Fourier domain) instead of the original standard signal basis set.

The DFT determines the Fourier coefficients by calculating the component of the signal in each of the Fourier basis vectors. Given the LS interpretation of the DFT, the transform assigns the weights (Fourier coefficients) in such a way as to minimise the square error between the original signal $y(n)$ and signal estimate $\hat{y}(n)$, which is formed by the linear combination of Fourier complex exponential basis functions weighted by the Fourier coefficients. We can think of DFT as assigning arbitrary weights initially for each basis function and then iteratively updating weights in such as way that best describes the original signal (minimising error at each step). Therefore, the optimal Fourier coefficients are those that provide the closest estimate to the original signal in the LS error sense when projected into the Fourier domain by the complex exponential basis set. It should be noted that DFT uses N mutually orthogonal basis vectors to provide an estimate of signal, minimising LS error, however in order for the error to reduce to zero ($\hat{y}(n) = y(n)$) and have an increase in accuracy, then $N \to \infty$ meaning that $\mathbf{F}_{DFT}$ must be a square matrix of infinite dimensions (i.e. when DFT becomes continuous-time Fourier transform).

Part c)

By treating the DFT as a least squares solution, allows us to implement an adaptive version using CLMS algorithm, where the Fourier coefficients are updated in an iterative manner and the input to CLMS at each time instant is the DFT basis function. This technique is known as the DFT-CLMS algorithm and the magnitude of the weight vector of filter gives the frequency spectrum at every time instant. The DFT-CLMS algorithm is implemented on the N-sample FM signal from Part 3.2 a) and the time-frequency diagram is shown in Figure 3.13. For this plot, each basis function input had a length of N (=1500), sampling frequency was set to 1500Hz, learning rate was $\mu = 1$ and initial weight values were set to zero.

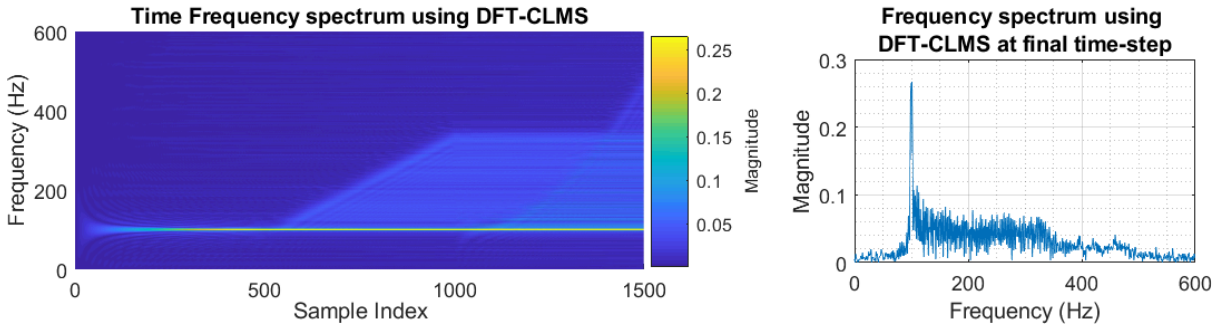


Figure 3.13: (Left) Time-frequency spectrum by DFT-CLMS algorithm for the FM signal. (Right) Frequency spectrum using DFT-CLMS at the final time-step, i.e. at n=1500.

As seen from Figure 3.13, the DFT-CLMS algorithm provides a faster adaptation (higher convergence rate) to changes of signal frequency when compared to the adaptive AR-spectrum analyser in Figure 3.12. The DFT-CLMS algorithm also has the advantage that it does not make assumptions about the signal spectrum (e.g. the number of spectral peaks). Also, as right-plot of Figure 3.13 shows, the resulting spectrum estimate is similar to that obtained for non-adaptive AR spectrum analyser in Figure 3.9, when a very high order ($M \gg 10$) is used. Additionally, for the region with constant frequency, DFT-CLMS converges faster to the true 100Hz frequency, with lower steady-state error, since the spectrum for this region has a much narrower spectral peak when compared to spectrum obtained from adaptive AR-spectrum analyser. However, as seen from both Figure 3.13 and 3.12, the AR-spectrum analyser outperforms the DFT-CLMS method for non-stationary parts of signal, i.e. linear and quadratic regions. Therefore this suggests that DFT-CLMS performs better for stationary signals while AR-spectrum analyser performs better for non-stationary signals. The reason for the degradation of performance of DFT-CLMS algorithm for non-stationary signal segments comes from the fact that once frequency estimates appear in the spectrum, they will remain in the time-frequency spectrum for the remaining time. As shown in Figure 3.13, frequency components are being accumulated over time and do not get updated, which means that at each time step the spectrum is an overlap of the spectrum at preceding time steps. This is highly evident at frequency spectrum at the last time-step of algorithm.
According to Widrow, B. et al. (1987, pp. 814–820) 'Fundamental Relations Between the LMS Algorithm and the DFT', in order for weight vectors $\mathbf{w}(n)$ to converge to DFT solution, the learning rate must be set to $\mu = 1$ and initial conditions to $\mathbf{w}(0) = 0$. In general, the weight update rule, where $\mathbf{x}(n)$ is input complex phasor and error term $e(n) = y(n) - \hat{y}(n)$, is $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e^*(n)\mathbf{x}(n)$. Following the reasoning from Widrow, B. et al, we can show that when $\mathbf{w}(0) = 0$, the equation above can be reduced to:

$$\mathbf{w}(n) = \mu \sum_{m=0}^{n-1} y^*(m)\mathbf{x}(m) \quad \text{and when } \mu = 1\text{:} \quad \mathbf{w}(n) = \sum_{m=0}^{n-1} y^*(m)\mathbf{x}(m) \tag{3.39}$$

This means that by setting $\mu = 1$ in Equation 3.39, the updated weight vector $\mathbf{w}(n)$ is proportional to the DFT of the previous $n-1$ samples of input data, verifying our observations from the time-frequency plot. This will not be an issue for stationary data, however, since frequency spectrum at each time step is the same. A possible solution to this issue will be a modification of the DFT-CLMS method to allow less sensitivity to past weight values. Using the idea from the Leaky LMS Algorithm is Section 2.1, we can introduce a leakage factor γ in the DFT-CLMS update equation, to produce Leaky DFT-CLMS. By choosing $\gamma = 0.05$, which was found empirically so that sensitivity for fast weight updates and signal distortion (introduced noise) are balanced, we obtain Figure . This shows that Leaky DFT-CLMS outperforms AR-spectrum analyser for both stationary and non-stationary segments, while having slightly wider spectral peak than the original DFT-CLMS algorithm for the constant frequency region.

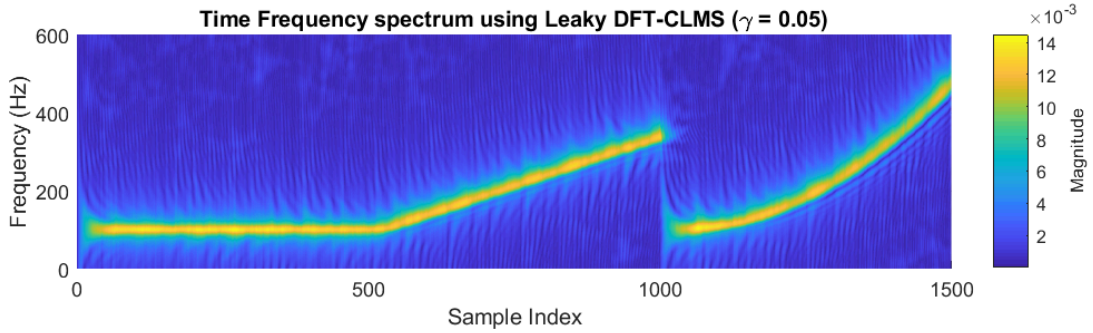


Figure 3.14: Time-frequency spectrum by Leaky DFT-CLMS algorithm for the FM signal, using $\gamma = 0.05$.

Part d)
The DFT-CLMS algorithm was also implemented for the EEG signal, from POz electrode, used in Part 1.2. Due to high computational burden, only a 1200-sample long segment of data was used, which was assumed stationary (that is why Leaky DFT-CLMS was not implemented). The time-frequency plot, obtained using the data segment `POz(1000:2199)`, is shown in Figure 3.15. From this plot, we can clearly identify the power-line interference at 50Hz and the fundamental and first harmonic of SSVEP at 13Hz and 26Hz respectively, but higher harmonics of SSVEP are not visible in spectrum. As we have seen in part c), the DFT-CLMS algorithm performs best for stationary signals. This is the reason why DFT-CLMS has good performance in spectrum estimation, since the EEG data is

highly stationary, especially within the short segment considered. Additionally, due to the stationary nature of data, the leaky DFT-CLMS does not provide any advantages but instead increases time of convergence and introduces noise artefacts to spectrum (discards information from data and distorts spectrum unless optimised γ is used).

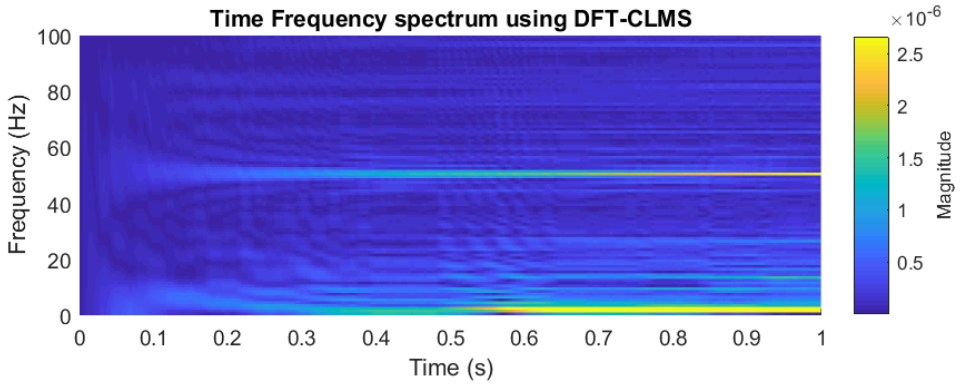


Figure 3.15: Time-frequency spectrum by DFT-CLMS algorithm for the EEG POz data.

4 From LMS to Deep Learning

4.1 Linear LMS Prediction for Non-stationary Data

As seen in Section 2, the LMS algorithm can be used for prediction of non-stationary time-series in an online fashion. For this task, the LMS algorithm is used to estimate the non-stationary time series shown in Figure 4.1, from its past values.

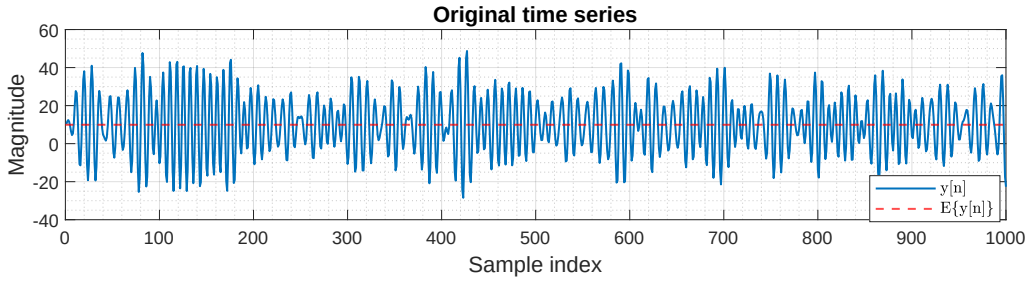


Figure 4.1: Original non-stationary time-series (blue line) with non-zero mean(broken red line).

The mean value of time-series was first removed, and then LMS algorithm was used (using learning rate $\mu = 1 \times 10^{-5}$) for one-step ahead prediction of data $y[n]$ by modelling the signal as an AR(4) process. Figure 4.2 provides the plot of the zero-mean signal against its AR(4) one-step ahead prediction. The LMS output is empirically found (through simulations of time-evolution of LMS coefficients) to have sufficiently converged to the original zero-mean data after approximately 200 time-steps. As shown in Figure 4.2, initially magnitude of predicted signal is zero and as time-steps increase, the output signal approaches closer and closer to the original signal (improvement in performance of LMS). During the first 200 time-steps (prior to convergence) the predicted signal is incorrect. However, for final time-steps the predicted signal behaves closely (in terms of variation trend), but not perfectly (in terms of actual values), to the original zero-mean data. Due to the slow convergence of LMS and the assumption of linearity of data, the prediction output results to high mean-square error (MSE) value and low Prediction Gain (R_p) value. Specifically, the estimated MSE was found to be 16.032dB, while the prediction gain $R_p = 10log_{10}\left(\sigma_{\hat{y}}^2/\sigma_e^2\right) = 5.195$ dB, which indicate that the prediction performance can be improved. It should be noted that the MSE quantifies how close the prediction is to the original signal and therefore the higher the MSE the poorer the performance of algorithm. In fact, during the first 200-samples, MSE= 21.27dB and for the remaining 800 samples MSE= 12.21dB. Finally, the prediction gain indicates the ratio of estimated variance of the original signal $\hat{\sigma}_y^2$ to the estimated variance σ_e^2 of the forward prediction error $e(n)$, thus the higher the R_p, the better the performance of algorithm. During the first 200-samples, $R_p = -1.93$dB and for the remaining 800 samples $R_p = 9.38$dB.

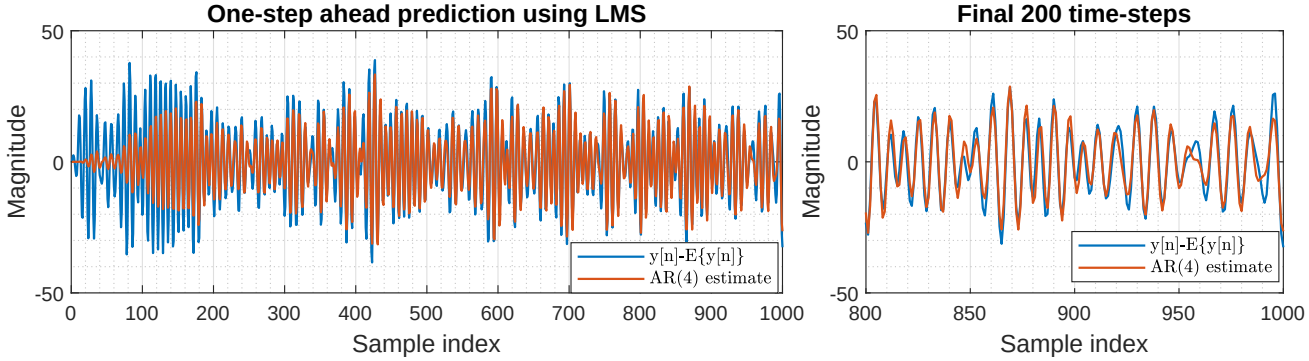


Figure 4.2: (Left) Zero-mean version of original signal (blue line) against its one-step ahead prediction (red line) using LMS. (Right) Zoomed-in version of the plot to illustrate LMS output for last 200 samples.

4.2 Non-linear LMS Prediction using Hyperbolic Tangent

The error in prediction can be minimised by adding a non-linearity (an activation function) to the output of LMS algorithm which makes the model more expressive. This is because the underlying process generating the data might be non-linear which is a possible reason for the high MSE and relatively low R_p. Figure 4.3 shows the one-step ahead prediction performed on the zero-mean signal using the `tanh` function, with unit scale, as a non-linearity to LMS output, using learning rate $\mu = 1 \times 10^{-5}$. This means that the signal estimate is given by $\hat{y}(n) = \tanh(\mathbf{w}^T\mathbf{x})$, where $\mathbf{w}$ is the vector of the non-linear AR(4) model coefficients and $\mathbf{x}$ is the LMS input vector, i.e. the 4 past values of signal $y(n)$. Additionally, the LMS update rule, using the gradient-descent method, can be shown to be:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)[1 - tanh^2(\mathbf{w}^T(n)\mathbf{x}(n))]\mathbf{x}(n) \tag{4.1}$$

which minimises the cost function $J(n) = (1/2)e^2(n) = (1/2)[y(n) - tanh(\mathbf{w}^T(n)\mathbf{x}(n))]$. Note that $[1 - \tanh^2(\mathbf{w}^T\mathbf{x})]$ is the derivative of the function $tanh(\mathbf{w}^T\mathbf{x})$ with respect to $\mathbf{w}^T\mathbf{x}$.

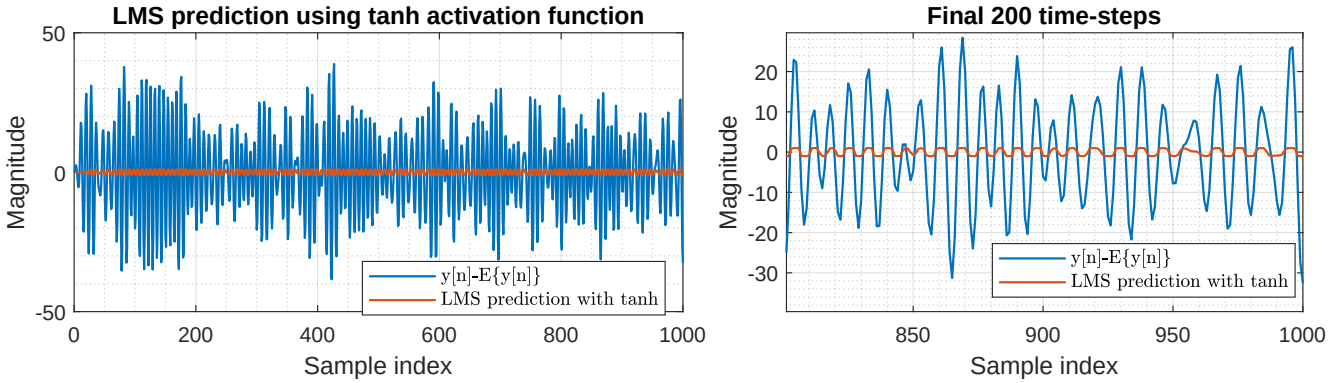


Figure 4.3: (Left) Zero-mean version of original signal (blue line) against its one-step ahead prediction (red line) using LMS with `tanh` activation function. (Right) Zoomed-in version of the plot to illustrate LMS output for the last 200 samples.

As shown in Figure 4.3, the `tanh` function with unit-scale fails to predict correctly the zero-mean signal. This is quantified by the very high MSE= 22.958dB and very low $R_p = -24.423$dB, which indicate the large deviation between estimated signal and the true zero-mean signal. The `tanh` function is an appropriate function to be used as an activation function for a dynamical perceptron model, since it is a non-linear, continuous, smooth function (continually differentiable with no discontinuities), allowing weight update equation to be expressed analytically, and is BIBO (Bounded-input Bounded-output) stable due to the saturation-type non-linearity of hyperbolic tangent (derivative of `tanh` at high input values is zero giving no update of weights). The `tanh` function provides this saturation-type non-linearity by mapping any input values on the real-axis, from $-\infty$ to $+\infty$ to an output between -1 and +1. The non-linearity provided by `tanh` allows the LMS algorithm to correctly capture the frequency variations in the original signal, however the algorithm is unable to capture the magnitude variations. Since output of `tanh` is limited in the range [-1,+1], any input values of $\mathbf{w}^T\mathbf{x}$ approximately > 3 or < -3 are saturating the output of the algorithm to a value of 1 or -1 respectively, and all other values range between 1 and -1. Since the prediction error is high enough due to differences in magnitudes, the LMS weights are updated to have saturation to minimise that error as much as possible. As a result, we have distortion of the LMS output signal by clipping, since its value is limited by the amplitude constraints of `tanh` function. In fact, the estimation using this method results in worse performance (higher MSE and lower R_p) compared to the standard linear LMS method in section 4.1. Therefore, the unit-scale `tanh` activation function is not an appropriate choice for this prediction problem, since the function's non-linearity cannot replicate signal's amplitude, but only the variations in trend. It should be noted that this activation function would work successfully if the original data was first normalised, so that its values lied between -1 and 1, but since now amplitudes are between [-38,38], we have very poor performance.

4.3 Scaled Non-linear Activation Function

The performance of the tanh-LMS algorithm can be improved by allowing a non-unit scale of `tanh` function, i.e. by changing its amplitude. It should be noted that by scaling the `tanh` function with a constant a, the function's output lies in the range $[-a, a]$. This means that using an appropriate value of a, the `tanh` function will be able to capture both amplitude and frequency variations of the original signal. To avoid clipping of output signal, the value of scale factor a should be at least greater than the maximum absolute value of the zero-mean signal, therefore $a \geq 38.8115$. Due to scaling factor a, the LMS output becomes $\hat{y}(n) = a\tanh(\mathbf{w}^T(n)\mathbf{x}(n)$ and thus the LMS update rule is given by:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)a[1 - tanh^2(\mathbf{w}^T(n)\mathbf{x}(n))]\mathbf{x}(n) \tag{4.2}$$

As Equation 4.2 indicates, the Learning rate is now effectively the product of μ and a which means that the higher the value of a the higher the overall learning rate. One possible option in this case, is to keep overall learning rate (μa) constant to the original value $\mu = 1 \times 10^{-5}$, by ignoring the effect of a in update equation (or changing μ in such manner to keep μa constant when a varies). Figure 4.4 shows how MSE and R_p of prediction change as a function of scale a when the overall learning rate is held constant. The optimal value of a in this case is found empirically to be ≈ 87.15, which is when MSE is minimum and R_p is maximum. As Figure 4.4 shows, the error decreases as a increases from 1 to the optimal value and then the starts increasing beyond the optimal value. The reason for increasing error is the high value of scale used which make cause amplitude of predicted signal to increase largely, specifically at the

local minimum following the location of maximum amplitude of original signal (observed from simulations).
On the other hand, the other option we have is to incorporate scale a in the update equation. In this case, we notice that if a is very high, the algorithm does not converge to the correct values (effects of high learning rates discussed in previous sections). In order to limit this effect we can reduce the value of μ to $1 \times 10^{-}7$ and vary a to find the optimal value. For this case, the optimal scale is $a = 92.5$. Figure 4.5 shows how MSE and R_p of prediction change as a function of scale a when the overall learning rate is not held constant. The shape of plots in Figure 4.5 is very similar to that of Figure 4.4. In this case, however, for values of a much larger than the optimal value (not shown on plot), the algorithm does not converge to the true signal, due to higher steady-state errors and for values of a much lower than the optimal value, the speed of convergence of algorithm is low which gives an additional error to clipping effect in the prediction.

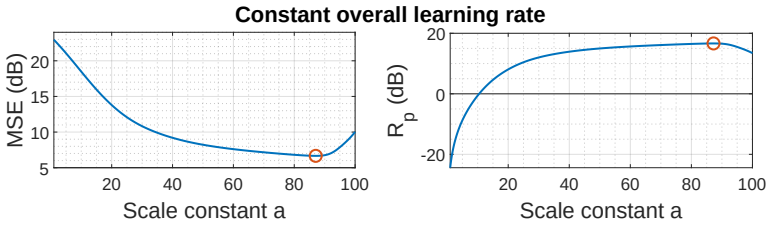


Figure 4.4: Plots showing how (Left) MSE and (Right) R_p change with scale constant a, when a is not incorporated into LMS update equation therefore having constant overall learning rate.

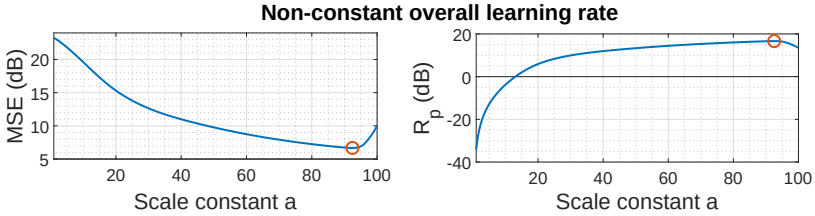


Figure 4.5: Plots showing how (Left) MSE and (Right) R_p change with scale constant a, when a is incorporated into LMS update equation therefore not having constant overall learning rate.

The method followed to find optimal a for the remaining parts of Section 4, is the second method described above. Figure 4.6 shows the one-step ahead prediction with the new proposed scaled activation function $a\dot{tanh}$, using $a = 92.5$ and $\mu = 1 \times 10^{-7}$, against the original zero-mean signal. For this method, the estimated values of MSE= 6.671dB and $R_p = 16.635$dB, which indicates the best performance compared to the other two algorithms considered so far in this section. By recalling that MSE= 16.032dB and $R_p = 5.195$dB for the standard linear LMS algorithm, we can conclude that scaled-tanh LMS has 2.40 times lower MSE and 3.20 times larger prediction gain compared to standard LMS algorithm. Also by comparing the final 200 time-steps of predicted signal from both methods (Figures 4.2 and 4.6), the scaled-tanh LMS method clearly replicates more closely both the magnitude and frequency variations of the signal. For example, considering the local minimum occurring at time-step of 991 in original signal, the scaled-tanh LMS estimate follows more closely the original signal than the estimate by standard LMS. This thus proves that the scaled-tanh LMS significantly outperforms the standard LMS in the prediction of non-linear and non-stationary signals. The only drawback for scaled-tanh LMS method is that the optimal value of scale a can only be determined after occurrence of signal.

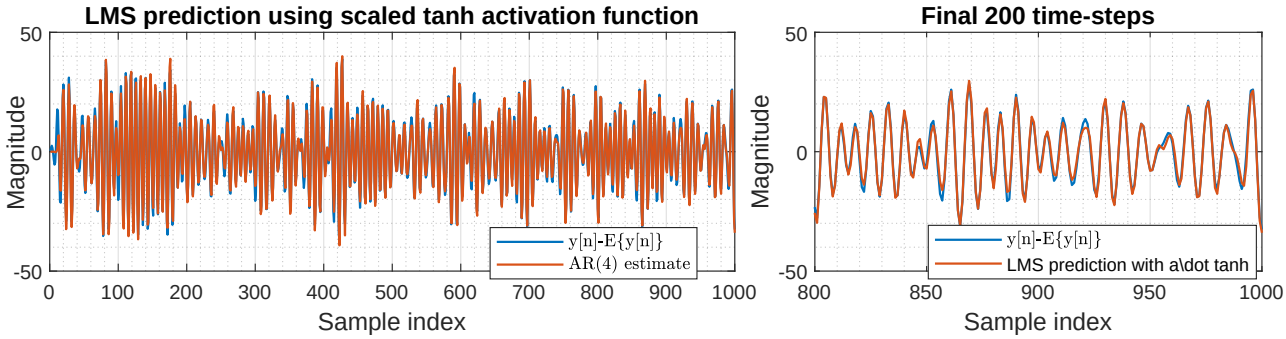


Figure 4.6: (Left) Zero-mean version of original signal (blue line) against its one-step ahead prediction (red line) using LMS with scaled `tanh` activation function. (Right) Zoomed-in version of the plot to illustrate LMS output for the last 200 samples.

4.4 Bias Addition in Non-linear Model

The original time-series from Figure 4.1 is now considered for the prediction task, which exhibits a non-zero mean value equal to 9.9244. To account for the mean automatically in the LMS, we add a bias term to the non-linear AR(4) model. The model in this case is described by $\hat{y}(n) = a \cdot tanh(\mathbf{w}^T(n)\mathbf{x}(n) + b)$, where b is the bias in the model. The bias is implemented by considering an 'augmented' input to the algorithm at each time step as $[1\ \mathbf{x}(n)]$, where $\mathbf{x}(n)$ are the 4 past values of time-series $y(n)$. This means that effectively we increase the number of coefficients by one and the bias b is computed at each time-step by $b(n) = 1(w_0(n)) = w_0(n)$, which accounts for the non-zero mean of data. Effectively the analytical expression of model is now given by $\hat{y}(n) = a \cdot tanh\left(b(n) + \sum_{i=1}^{4} w_i(n)y(n-i)\right)$. Figure

4.7 shows a comparison between the original non-zero mean signal and the non-linear one-step ahead prediction using the method with bias. It should be noted that the LMS update rule used is the same as that given by Equation 4.2, with the difference that vector $\mathbf{w}(n)$ and $\mathbf{x}(n)$ are 5×1 column vectors. It should be also noted that for this method, learning rate $\mu = 10^{-7}$ was used giving an optimal scale of 75.695.

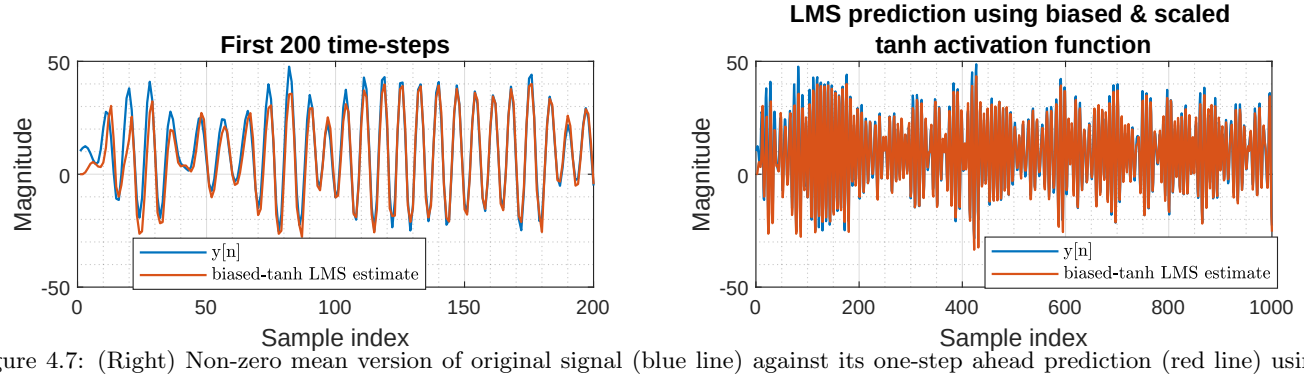


Figure 4.7: (Right) Non-zero mean version of original signal (blue line) against its one-step ahead prediction (red line) using LMS with biased scaled `tanh` activation function. (Left) Zoomed-in version of the plot to illustrate LMS output for the first 200 samples.

As Figure 4.7 shows, the biased-scaled-tanh LMS algorithm can perform considerably well in the prediction of non-zero mean data, giving an MSE= 10.173dB and R_p = 13.414dB, which are 1.58 times lower than MSE and 2.58 times higher than R_p values obtained using the standard linear LMS algorithm respectively. This signifies the improvement in performance of standard LMS algorithm with the introduction of non-linear tanh function, scale and bias. On the other hand, this method's performance is worse than that of scaled-tanh LMS for zero-mean data, since MSE increases by a factor of 1.52 and R_p decreases by a factor of 1.24. This is due to the additional task of the algorithm to estimate the DC offset bias in the signal. It should be also noted that for this reason the algorithm converges at a lower speed to the true signal compared to the unbiased algorithm implemented on the zero-mean signal. This is identified from Figures 4.7 and 4.6 during the first 200 samples, where the biased LMS algorithm takes more time-steps to reach the signal values, which is the reason for the higher MSE and lower R_p values.
By simulating the time evolution of the LMS weights, shown in Figure 4.8, we can observe that the biased biased algorithm for the non-zero mean data converges after approximately 150 time-steps. The MSE prior to convergence is found to be 15.448dB and after convergence is 6.276dB, while R_p prior to convergence is 10.658dB and after convergence is 16.512dB. Despite the fact that the predicted signal is very close to the original time-series, as evidenced from the low MSE and high R_p, we can identify from Figure 4.7 that the algorithm cannot completely capture the largest amplitudes of the original data, although works better for lowe amplitudes and captures all frequency variations in signal. This suggests that the implementation of bias in the algorithm cannot capture completely the mean value of data (i.e. effect of bias in algorithm is not significant). In fact, the unbiased-scaled-tanh LMS algorithm from Part 4.3, was used for the prediction of the non-zero mean data, yielding an output very close to that of the biased LMS algorithm with MSE= 10.254dB (0.792% absolute difference) and R_p = 13.347dB (0.495% absolute difference). Additionally, the mean value of original data is 9.9244, while for biased-estimate signal is 9.2571 and for non-biased-estimate signal is 9.2336, suggesting mean value is not sufficiently captured. The low impact of bias on estimation is also seen from the fact that the optimal scale required using the bias (a_{opt} = 75.695) is approximately the same to method not using bias (a_{opt} = 75.675). Another observation was made regarding the coefficient w_0(=bias) in Figure 4.8, which does not converge completely to their optimal value, but increases slightly after each time-step, while other coefficients have shown sufficient convergence. Finally, by considering the time-evolution of coefficients for the biased and unbiased algorithms, it was observed that all coefficients w_1 to w_4, have the same convergence speed, although coefficients of biased-method converge to values with slightly lower amplitude.

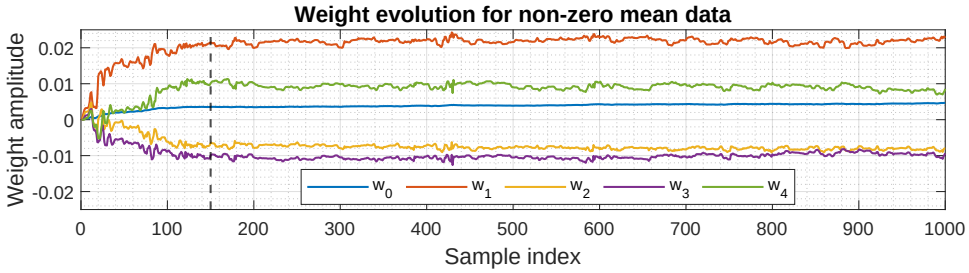


Figure 4.8: Time evolution of the 5 coefficients of biased-scaled-tanh LMS algorithm, denoted by w_i. The broken black line denotes the approximate time of convergence.

These observations can be explained from the fact that the input data (past 4 signal values) are not normalised and the bias is applied before the non-linearity. Since the input value of 1 is much lower than signal peak amplitude values, the input bias will not have any significant effect in performance, verifying our observations. Therefore, in order to see a significant increase in performance due to bias implementation, we could potentially normalise data first, so the value of 1 in the augmented input is comparable in magnitude to that of input data. An alternative option to the normalisation of input data, is to use the augmented input to algorithm as $[\beta\ \mathbf{x}]$, where we can vary parameter β to any value. It was empirically found for this particular example that when the value of β is close to value of 10, we

obtain an MSE value equal to 8.963dB and $R_p = 14.346$dB, where mean value of estimated data is 9.7948. The MSE and R_p values for different values of β are shown in Table 12. It was also observed that for $beta > 1$, coefficients w_1 to w_4 of biased method are becoming much lower in amplitude compared to those of method where bias is not implemented.

Table 12: MSE and R_p for different values of parameter β. For $\beta = 0$, we have the unbiased-scaled-tanh LMS and for $\beta = 1$, we have the biased-scaled-tanh LMS given by the task.

β	**0**	**1**	**2**	**5**	**10**	**15**	**20**	**25**	**30**	**50**
MSE (dB)	10.254	10.173	9.966	9.345	9.272	9.527	9.827	10.152	10.521	11.856
R_p (dB)	13.348	13.414	13.580	14.051	14.027	13.754	13.455	13.140	12.779	11.450

4.5 Pre-training of Model Coefficients

The slow convergence of the biased-scaled-tanh LMS algorithm can be explained from the fact that only a single weight update is performed at each time-step. To resolve this issue, we can pre-train the LMS weights by over-fitting to a small number of signal samples. Specifically, by starting with $\mathbf{w}(0) = \mathbf{0}$ and using 100 iterations (epochs) we fit the model to the first 20 samples of signal (training data set) to yield $\mathbf{w}_{init}$, which is then used as an initialisation for LMS weights for the prediction of the entire non-zero mean time-series. Figure 4.9 shows the plot of the original signal (non-zero mean) and the non-linear one-step ahead prediction produced with bias and pre-trained weights, using learning rate $\mu = 10^{-7}$ and optimal scale $a = 67.95$ (obtained using method described in Part 4.3).

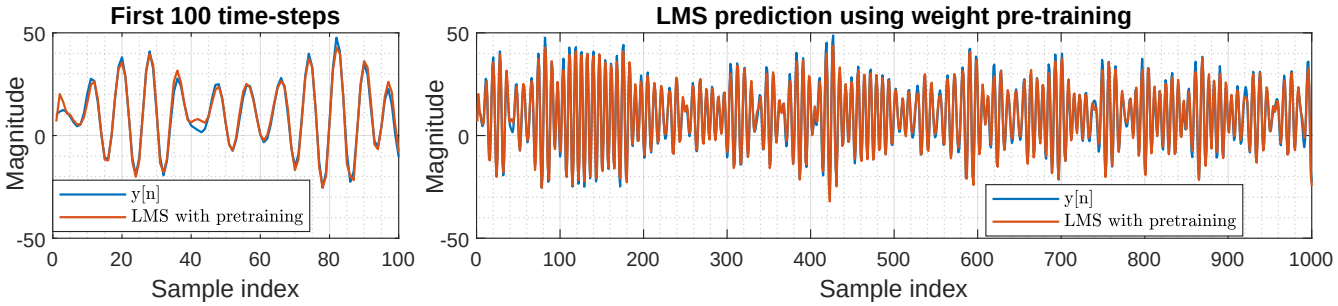


Figure 4.9: (Right) Non-zero mean version of original signal (blue line) against its one-step ahead prediction (red line) using LMS with biased scaled tanh activation function and pre-training of weights. (Left) Zoomed-in version of the plot to illustrate LMS output for the first 100 samples.

As Figure 4.9 shows, pre-training of algorithm weights significantly improves the converge speed of the algorithm, since the convergence occurs within less than 5 time-steps. Additionally, this method improves largely the performance of the algorithm yielding an MSE value of 6.380dB (1.59 times lower than before) and an R_p value of 16.979dB (1.27 times higher than before).By observing also the mean value of estimated data, we obtain a value equal to 10.1025, with 1.7944% absolute difference to true value (9.9244), while without the pre-training approach the mean obtained was 9.2571, with 6.7233% absolute difference to true value. By simulating the time-evolution of coefficient w_0 of algorithm, which represents the bias, we can identify that the bias using pre-training method converges to a much higher value (0.1003) compared to previous method (0.0047), which is at least 5 times larger in amplitude compared to the remaining coefficients of algorithm. This shows that the effect of bias is more pronounced using this method. This is the reason why the mean of estimate is closer to the true mean and also the reason for the lower optimal scale required (with pre-training $a_{opt} = 67.95$ and without pre-training $a_{opt} = 75.695$) since when mean is successfully captured by the bias term a lower scale will be needed. This observation also suggests that for the method not utilising pre-training of weights, the coefficient for bias has not yet converged to the true value after 100 samples.

4.6 Backpropagation Algorithm

For modelling of real-life problems, a simple activation function for a single perceptron, like `tanh` used in previous parts, is usually not expressive enough, due to highly non-linear mapping of input to output. To make the model more expressive for real-life problems, we can arrange many dynamical perceptron models in order to create a neural network. For nonlinear prediction, we are usually interested in networks consisting of multiple neuron layers, where the output of one layer is fed as input to the next layer, thus creating a deep feed-forward neural network, an example of which is shown in Figure 4.10. Each perceptron is represented as a node in the diagram and arrows represent weights connecting the previous to the next layer. Update or fine tuning of the network weights (and biases) are updated based on the gradient of the cost function $J(n) = (1/2)e(n) = (1/2)(y(n) - \hat{y}(n))$, where y is the true signal and $\hat{y}$ is network's output. The algorithm used for the supervised learning of a neural network is called Backpropagation algorithm, which gives an expression of the rate of change of cost function (obtained from the previous epoch) with respect to weights and biases in network. In other words, using backpropagation, we are basically feeding backwards the error in the network from the network output layer-by-layer, allowing tuning of weights in such a way to obtain a lower error in the next epoch of training. Therefore, backpropagation algorithm aims to adjust weight and bias values in each layer of network to minimise the cost function, using a gradient-descent approach. The direction of propagation of error is opposite (backwards) to the direction that input data flows in the network.

Backpropagation algorithm repeats until no further weight updates are made or a certain threshold in cost function is reached or when maximum number of epochs has been reached.

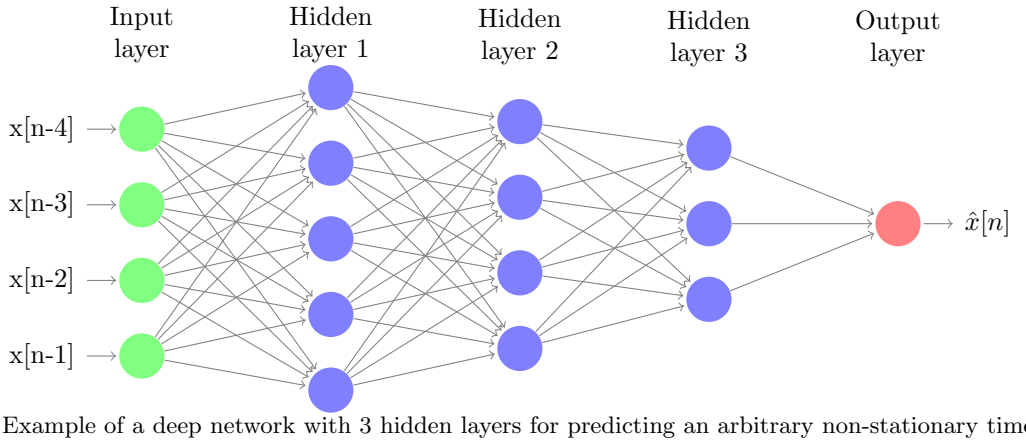


Figure 4.10: Example of a deep network with 3 hidden layers for predicting an arbitrary non-stationary time-series x[n].

By considering a fully-connected Multilayer Perceptron (MLP), with an arbitrary number of hidden layers and all perceptrons having the same activation function σ, we can denote the weight for connection between the k^{th} neuron in the $(L\text{-}1)^{\text{th}}$ layer and the j^{th} neuron in the L^{th} layer as w_{jk}^L, the bias input of the j^{th} neuron in L^{th} layer as b_j^L and its output as g_j^L. Therefore, the output of j^{th} neuron in L^{th} layer is given by:

$$g_j^L = \sigma\left(b_j^L + \sum_k w_{jk}^L g_j^{L-1}\right) \tag{4.3}$$

This can also be expressed in vector form, considering all neurons in L^{th} layer as:

$$\mathbf{g}^L = \sigma\left(\mathbf{b}^L + \mathbf{w}^L\mathbf{g}^{L-1}\right) = \sigma(\mathbf{z}^L) \quad \text{where} \quad \mathbf{z}^L = \mathbf{b}^L + \mathbf{w}^L\mathbf{g}^{L-1} \tag{4.4}$$

Using the vector form given by Equation 4.4, the weight and bias update equations for all connections to the L^{th} layer are:

- $\mathbf{w}^L(n+1) = \mathbf{w}^L(n) - \mu\dfrac{\partial J}{\partial \mathbf{w}^L}$ (4.5)
- $\mathbf{b}^L(n+1) = \mathbf{b}^L(n) - \mu\dfrac{\partial J}{\partial \mathbf{b}^L}$ (4.6)

Using the chain rule, we can show that, if the MLP has M layers, where output layer is the M^{th} layer and input layer is 0^{th} layer, then the weight and bias update equations for all connections to output layer are given by:

- $\mathbf{w}^M(n+1) = \mathbf{w}^M(n) + \mu e(n)\sigma'(\mathbf{z}^M)\mathbf{g}^{M-1} = \mathbf{w}^M(n) + \mu\boldsymbol{\delta}^M\mathbf{g}^{M-1}$ where $\boldsymbol{\delta}^M = e(n)\sigma'(\mathbf{z}^M)$ (4.7)
- $\mathbf{b}^M(n+1) = \mathbf{b}^M(n) + \mu e(n)\sigma'(\mathbf{z}^M) = \mathbf{b}^M(n) + \mu\boldsymbol{\delta}^M$ (4.8)

These equations indicate the need for a smooth and continuous (differentiable) activation function σ. Considering now the update equation for all hidden layers (and modelling input layer as perceptrons with same input as output), we can use the chain rule again to find weight and bias update rules for the preceding hidden layer to output layer, i.e. $(M-1)^{\text{th}}$ layer:

- $\mathbf{w}^{M-1}(n+1) = \mathbf{w}^{M-1}(n) + \mu e(n)\sigma'(\mathbf{z}^M)\mathbf{w}^M(n)\sigma'(\mathbf{z}^{M-1})\mathbf{g}^{M-2} = \mathbf{w}^{M-1}(n) + \mu\boldsymbol{\delta}^{M-1}\mathbf{g}^{M-2}$ (4.9)
- $\mathbf{b}^{M-1}(n+1) = \mathbf{b}^{M-1}(n) + \mu e(n)\sigma'(\mathbf{z}^M)\mathbf{w}^M(n)\sigma'(\mathbf{z}^{M-1}) = \mathbf{b}^{M-1}(n) + \mu\boldsymbol{\delta}^{M-1}$ (4.10)

  where $\boldsymbol{\delta}^{M-1} = \sigma'(\mathbf{z}^{M-1})\boldsymbol{\delta}^M\mathbf{w}^M(n)$

Therefore considering any pair of hidden layers (e.g. k^{th} and $(k-1)^{\text{th}}$ layers) in the network, the generalised delta rule specifies that:

$$\boldsymbol{\delta}^{k-1} = \sigma'(\mathbf{z}^{k-1})\boldsymbol{\delta}^k\mathbf{w}^k(n) \tag{4.11}$$

- $\mathbf{w}^{k-1}(n+1) = \mathbf{w}^{k-1}(n) + \mu\boldsymbol{\delta}^{k-1}\mathbf{g}^{k-2}$ (4.12)
- $\mathbf{b}^{k-1}(n+1) = \mathbf{b}^{k-1}(n) + \mu\boldsymbol{\delta}^{k-1}$ (4.13)

Therefore, this means that the error is propagated first through the output layer from the output of the network to obtain $\boldsymbol{\delta}^M$, and then each time the error is propagated one layer back to update weights and biases of all hidden layers step-by-step, using Equation 4.11. Considering as an example the fully-connected network in Figure 4.10, the update rule for weight connections between hidden layer 2 and 3, using the same notation, is given by:

$$\mathbf{w}^3(n+1) = \mathbf{w}^3(n) + \mu\boldsymbol{\delta}^3\mathbf{g}^2 = \mathbf{w}^3(n) + \mu\sigma'(\mathbf{z}^3)e(n)\sigma'(\mathbf{z}^4)\mathbf{w}^4(n)\mathbf{g}^2 \tag{4.14}$$

and for weight connections between hidden layer 1 and 2:

$$\mathbf{w}^2(n+1) = \mathbf{w}^2(n) + \mu\boldsymbol{\delta}^2\mathbf{g}^1 = \mathbf{w}^2(n) + \mu\sigma'(\mathbf{z}^2)\sigma'(\mathbf{z}^3)e(n)\sigma'(\mathbf{z}^4)\mathbf{w}^4(n)\mathbf{w}^3(n)\mathbf{g}^1 \tag{4.15}$$

During the training process of a deep network used for a prediction task, the Backpropagation Training algorithm is performed in 7 steps. (1)The weights and biases of the network are initialised to small random values. (2)A number of past values of data (depending on model order) to be estimated is used as input to input layer of network. (3)The input is then fed-forward through the network where the output of each perceptron is given by Equation 4.3. (4)The prediction error is computed between network output and true signal. (5) The delta's for the output layer are computed according to Equation 4.7. (6)The delta's for all preceding layers are computed by propagating the error backwards using Equation 4.11. (7)The weights and biases are updated according to generalised delta rule Equations 4.12 and 4.13. After completion of step 7, the Backpropagation algorithm repeats from step 2 and stops until a condition is met or the maximum number of iterations has been reached, where training of deep network is completed.

4.7 Deep Network Performance for Prediction

Deep Neural networks can be a useful tool for prediction of highly non-linear signals. For this task, the signal of interest consists of a highly non-linear combination of varying amplitude (some with exponentially-decaying amplitude) and frequency sinusoids using unknown non-linear function ϕ and the signal is then corrupted using white Gaussian noise of varying powers. We can apply the universal function approximation property of a Neural Network (NN) to give an approximation to ϕ in order for the output of the NN to give an estimate of true signal. The same procedure is also repeated using a simple dynamical perceptron, with linear and non-linear (`tanh`) activation functions. In order for the NN to have the ability to approximate any continuous function, the activation function used in all hidden layers must be non-linear. Specifically for this case the activation function used was the rectified linear unit (ReLU) and not huberbolic tangent function. The reason for this is that ReLU is widely used for regression problems, since it has lower computational complexity than tanh for gradient computations (desirable if we have many layers), is more robust to initialisation of weights also does not suffer from the vanishing gradient problem present in logistic functions (gradient values are close to 0 for large input values so network may not converge). The desired signal to be estimated $y[n]$ and the input (feature) vector $\mathbf{x}[n]$ used as input to both NN and dynamical perceptron model, are shown in Figure 4.11.

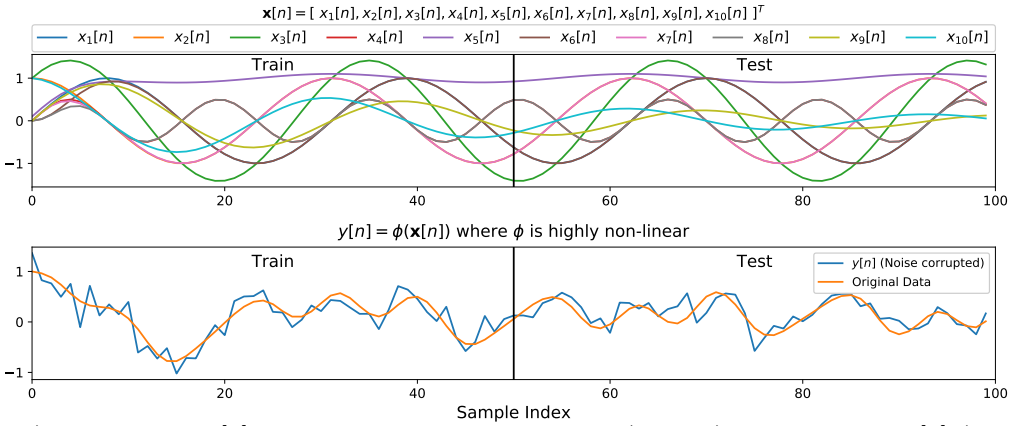


Figure 4.11: (Top) Input vector $\mathbf{x}[n]$ to NN and perceptron models. (Bottom) Desired Signal $y[n]$ (noise-corrupted) and noise-free data.

For this prediction task, the deep network was trained using 4 hidden layers, 20000 epochs, learning rate $\mu = 0.01$ and noise power equal to 0.05. The signal estimates using the three methods are shown in Figure 4.12. As Figure 4.12 shows, the methods using simple perceptron models (linear and `tanh`) can only roughly predict the general trend in data, but do not succeed in capturing the complete signal behaviour, e.g. faster variations in signal, mainly due to limited size of input vector. It should be noted that the estimates for both methods produce very similar estimates, which is due to the fact that for tanh function, any inputs between -0.5 and 0.5, result in approximately a linear mapping to the output (with gradient ≈ 1). Since input vector is non-linear, the linear (or approximately linear for most input values) combination of its independent components can be used to approximate the signal. On the other hand, the output of the deep network is approximately the local average of true signal and thus replicating the signal's trend, but does not succeed in capturing the full signal behaviour (lower amplitude variations) and results in wide flat output regions.

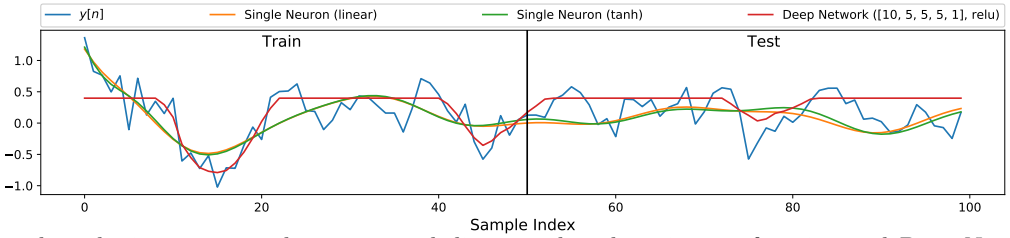


Figure 4.12: Signal prediction using single neuron with linear and tanh activation function and Deep Network using ReLU activation function. Noise power is equal to 0.05.

The learning curves for the three methods, based on the mean squared error (squared L2 norm) between the actual signal and the predicted signal, are shown in Figure 4.13. To allow quantitative comparisons for the performance of the three methods, the convergence time as well as the minimum and steady-state values of loss function were obtained and are shown in Table 13. It should be noted that the minimum error is the most important metric for the performance of each method, since we could potentially stop training once this point is reached. The results in Table 13, show that the deep network outperforms the simple dynamical perceptron models since it achieves a lower minimum prediction error for test signal (higher generalisation capability) and a lower steady-state error for both train and test signals. This increase in test performance of deep network can be explained from the fact that the network has higher degrees of freedom (higher expreesive power) to model signal more accurately and the model is more expressive of the highly nonlinear mapping of input to output. Comparing the two simple dynamical perceptron models, as results in Table 13 suggest, the neuron with tanh activation function performs better (lower minimum and steady-state error) than linear activation function since non-linearity introduced by tanh allows for sufficient non-linear mapping of neuron input to output. Additionally, the deep network method has the lower convergence speed due to the much higher number of parameters that are tuned by the backpropagation algorithm (higher computational complexity). The convergence

time for the non-linear perceptron is lower than that of liner perceptron due to the non-lineary provided by tanh function which is more expressive to model the data.

Table 13: Approximate convergence time and MSE errors for the three methods, using noise power equal to 0.05. Estimated convergence time is found when the test loss has converged to a steady value.

Method	Convergence Time (epochs)	Minimum error (Test)	Steady-state error (Train)	Steady-state error (Test)
Single Neuron (linear)	7500	0.0881	0.0941	0.0939
Single Neuron (tanh)	5000	0.0774	0.0923	0.0897
Deep Network	12500	0.0693	0.0754	0.0846

By close inspection of Figure 4.13, we can observe that in all learning curves of the three method the plot of Test (or validation) loss decreases until a minimum value and then starts increasing again. For the deep network test loss plot specifically, we can notice a local oscillation during initial epochs, where despite the fact that MSE is minimised, the deep network changes its parameters since train loss is still not minimised. This indicates that all methods suffer with over-fitting, which means that the models become well adapted to the training data, including the random fluctuations due to noise. As a result, the models lose their ability to generalize to the new test data, which causes an increase in error. This might be due to the fact that the models were trained for too long (convergence time much smaller than 20,000 epochs). Ideally, we do not want complete convergence in training data, so that our models can generalise better on test data.

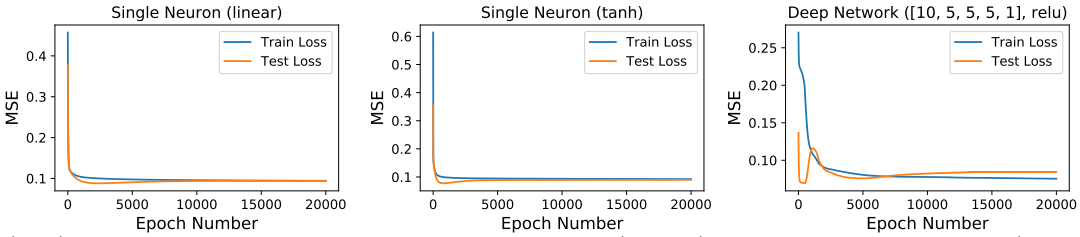


Figure 4.13: (Left) Learning curves plots for Linear single perceptron, (Middle) Tanh single perceptron and (Right) Deep Neural Network. Noise power is equal to 0.05.

4.8 Effect of Noise Power

The same procedure is repeated using the three methods described in Part 4.7 for different values of noise power, specifically for values 0.01 and 0.5, in order to analyse their performance for the same prediction task. For each method and noise power, the convergence time, minimum error and steady-state error was obtain and is illustrated in Table 14. The signal estimate for each method, using noise power equal to 0.01, is shown on Figure 4.14 and the corresponding learning curves for training and test (validation) cost/loss function (MSE) against epoch number on Figure 4.15.

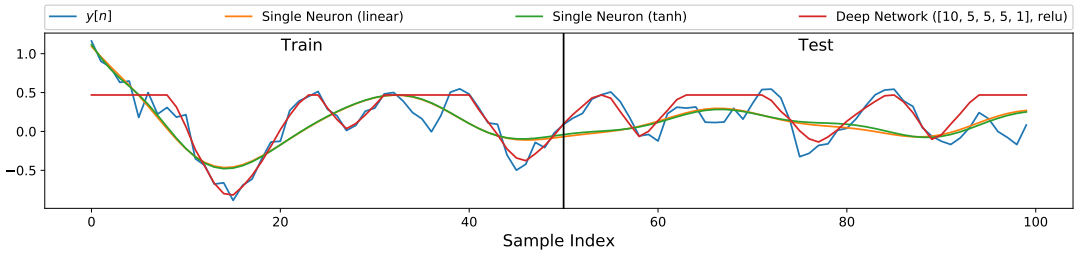


Figure 4.14: Signal prediction using single neuron with linear and tanh activation function and Deep Network using ReLU activation function. Noise power is equal to 0.01.

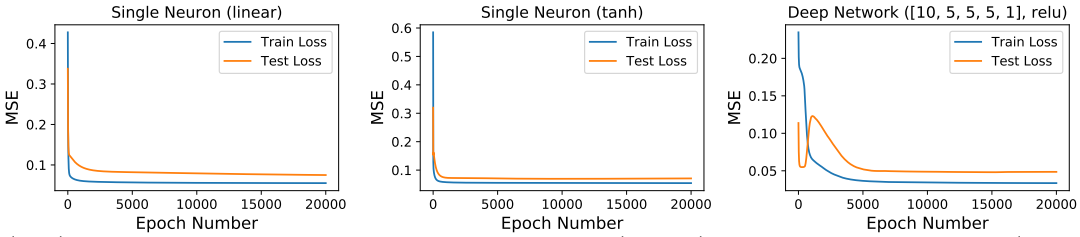


Figure 4.15: (Left) Learning curves plots for Linear single perceptron, (Middle) Tanh single perceptron and (Right) Deep Neural Network. Noise power is equal to 0.01.

As Figures 4.14 and 4.15 show, the three methods show improved performance compared to the case when noise power is 0.05, due to lower minimum errors and steady-state errors achieved, shown in Table 14 compared to those in Table 13. Having lower noise power in signal also prevents overfitting of models to training data as minimum error occurs very close to time of convergence and reduces algorithms' convergence time. This is also detected from the signal estimates in Figure 4.14, where for simple perceptron models, both linear and non-linear, output signal follows closely the general trend in data and larger amplitude variations but still cannot predict lower-scale variations of signal as in Figure 4.12. On the other hand, the deep network can model more closely the signal for both the train and test data sets, with lower convergence time and lower minimum error without overfitting occuring. Its performance is again the best of the three methods due to its higher expressive power and generalisation capability compared to a single perceptron. Despite observing again a local oscillation in test learning curve, the plot successfully converges to a lower steady-state error than when noise power was higher. The signal estimate for each method, using higher noise power

equal to 0.5, is shown on Figure 4.16 and the corresponding learning curves for training and test (validation) cost/loss function (MSE) against epoch number on Figure 4.17, where we can identify significant differences compared to plots for noise power equal to 0.01.

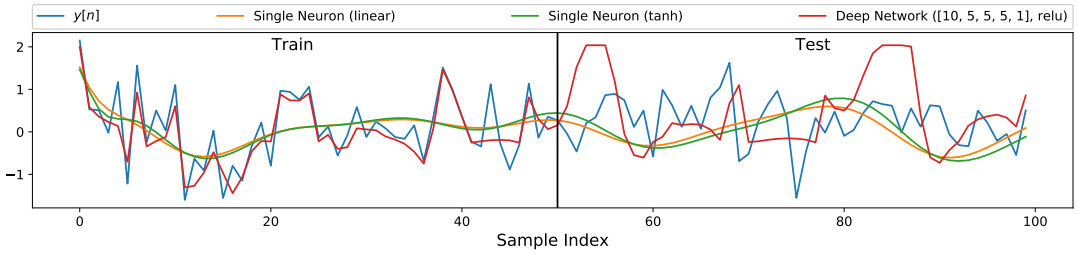


Figure 4.16: Signal prediction using single neuron with linear and tanh activation function and Deep Network using ReLU activation function. Noise power is equal to 0.5.

As Figure 4.16 shows, the deep neural network over-fits significantly the data including random outliers due to noise, resulting in poor performance on test data. On the other hand, the estimates obtained from simple perceptron models are less affected by noise and still follow the general trend of data although less weakly than for lower noise powers and are also affected by over-fitting. It is also observed that signal estimates from the single perceptrons can estimate more poorly the larger variations in data. Over-fitting is identified in Figure 4.17 for single perceptrons from the continuous decrease in train loss function and from the fact that test loss function decreases to a minimum value and starts increasing again. Figure 4.17 also shows over-fitting in neural network where test error massively increases after minimum error is reached. It should be noted that all three methods do no show clear convergence in test data (or training data). Despite the fact that the neural network obtains the lowest minimum error of the three methods, after 20,000 epochs both simple perceptron models perform better (with approximately half the test error) and therefore their outputs is more appropriate for the prediction task. Finally, the additional expressive power of the non-linearity of simple perceptron leads to more profound effects due to over-fitting (despite its minimum error being lower than linear model) which constitutes linear perceptron output to have the lowest error after 20,000 epochs. This shows that as the noise in signal increases, the error due to over-fitting also increases, and is higher for models with higher complexity (i.e. neural networks).

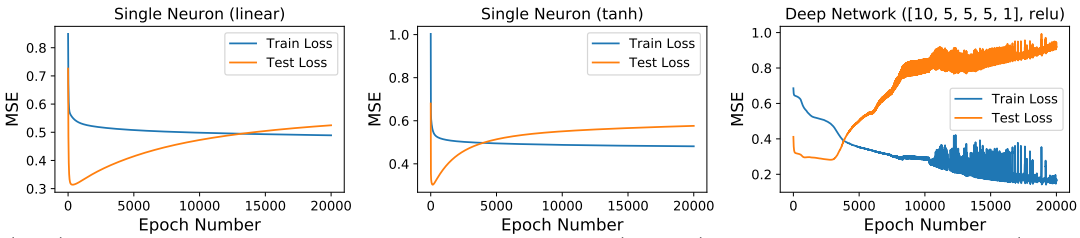


Figure 4.17: (Left) Learning curves plots for Linear single perceptron, (Middle) Tanh single perceptron and (Right) Deep Neural Network. Noise power is equal to 0.5.

Table 14: Approximate convergence times and MSE errors for the three methods, using different noise powers.

Noise Power	Method	Convergence Time (epochs)	Minimum error (Test)	Steady-state error (Train)	Steady-state error (Test)
0.01	**Single Neuron (linear)**	5000	0.0751	0.0551	0.0751
	Single Neuron (tanh)	2500	0.0694	0.0592	0.0707
	Deep Network	6000	0.0481	0.0336	0.0486
0.5	**Single Neuron (linear)**	-	0.3139	0.4892	0.5248
	Single Neuron (tanh)	-	0.3031	0.4814	0.5762
	Deep Network	-	0.2820	0.1696	0.9182

As we have seen, deep networks obtain the lowest minimum test error in prediction task due to high expressive power and generalisation capability. This makes them ideal for low-noise highly-nonlinear signals, whereas simple perceptron models perform better than networks in high-noise signals. Despite these advantages, there are several drawbacks on using neural networks for prediction. Firstly, the performance of the network depends on input feature vectors, which must be properly chosen for the underlying signal to be estimated. Neural networks are inherently highly computationally intensive (due to curse of dimensionality) and have generally lower convergence speed (from backpropagation algorithm) compared to simple perceptron models. Another drawback is the "black box" nature of the neural network output, since we can not easily determine which input variables contribute most to output because we are unable to interpret easily the weights (or biases) of neural connections. A significant disadvantage of neural networks is that they are prone to over-fitting of the training data set, especially in the case of small data sets, which results in poor performance in test data sets. This indicates that there is a trade-off between depth of network (comparing deep network to shallower networks or simple perceptron models) and speed of response to random outliers and anomalies due to noise in the signal. Decreasing depth, however, limits ability of nwetwork to model complex non-linear data. In other words, the higher the depth of network (number of hidden layers) the more significant are the test errors due to over-fitting. However, there are many techniques available that can prevent over-fitting. For example, the regularization technique called dropout can be used, where random perceptrons are ignored during network training. An additional method to prevent over-fitting, is to artificially add noise to network inputs or network weights (and biases) to make netwrok more robust, with improved generalisation and faster learning.