Imperial College London

Spectral Estimation & Advanced Signal Processing

Author:
Thomas Lim

CID:
00687826

April 2015

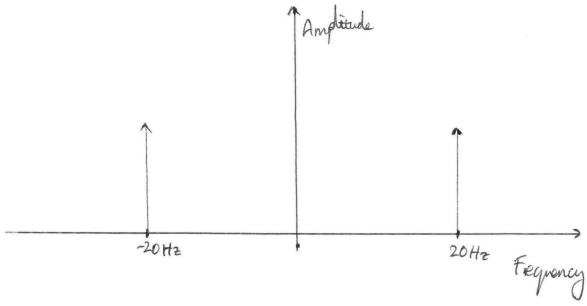
Contents

This report was typeset using LaTeX. Various open source `MATLAB` toolboxes were used to aide figure rendering and exporting: `matlab2tikz`, `export_figure`, `distinguisable_colours` and `tightfig`.

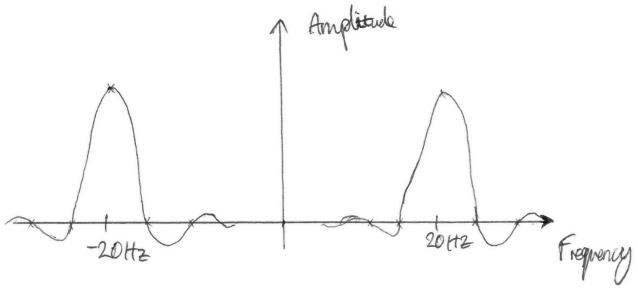
1 Non-parametric Spectrum Estimation

1.1 Discrete Fourier Transform Basics

1.1.a Ideal Fourier Spectrum



(a) *20Hz sine wave in the theoretical continuous frequency DTFT*

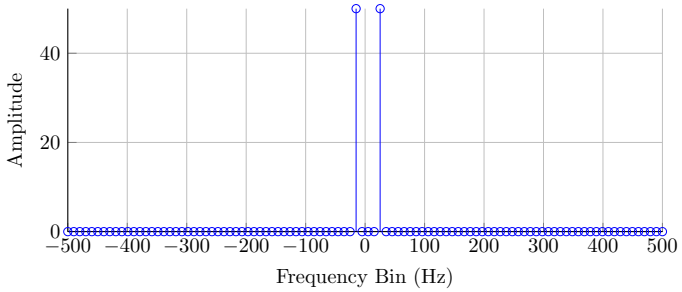


(b) *20Hz sine wave windowed (with an arbitrary window)*

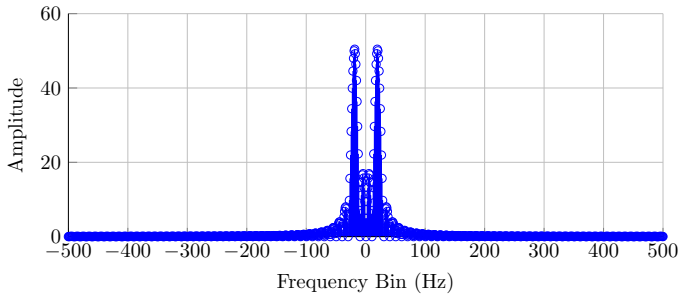
Figure 1.1: *DFTs of 20Hz Sine Wave, with* $F_s = 1000Hz$

Figure **??** shows a sine wave sampled in the theoretical continuous frequency DTFT, and a windowed sine wave. The result of the windowing is that we no longer see a clean delta in the frequency domain, but instead a sinc function, centred around 20 and -20 Hz.

1.1.b Sampled DFT Signals



(a) *20 Hz sine wave, with* $N = 100$ *points sampled on the* $K = 100$ *DFT*



(b) *20 Hz sine wave, with* $N = 100$ *points sampled on the* $K = 1000$ *DFT*

Figure 1.2: *DFTs of 20Hz Sine Wave, with* $F_s = 1000Hz$

Figure **??** shows a sine wave of 20 Hz with length N=100 samples, sampled with $K = 100$ (Figure 1.2a) and $K = 1000$ (Figure 1.2b). They resemble the plots in Section 1.1.a. Figure 1.2b shows the absolute value of each bin, but still represents a sinc function as sketched. When we take the DFT for $K = 1000$ but with $N = 100$ samples, we are zero padding the signal, which is equivalent to applying a rectangular gate to an otherwise infinite sinusoid (it appeared infinite in the $K = 100$ sample since it was the length of the signal). This rectangular gating causes sinc function we we observe in the Frequency domain.

1.1.c Incoherent Sampling

Figure 1.3 shows a 24Hz sine wave sampled with $F_s = 1kHz$, 100 samples long. 24 kHz does not fit in to any bins for a $K = 100$ point DFT. The frequency resolution is defined by $\triangle f = \frac{F_s}{N}$, which in this case results in $\triangle f = 10Hz$. Thus we are unable to see 24Hz exactly. In order to increase the resolution, we can either take more samples (which takes more time), or increase the sampling frequency.

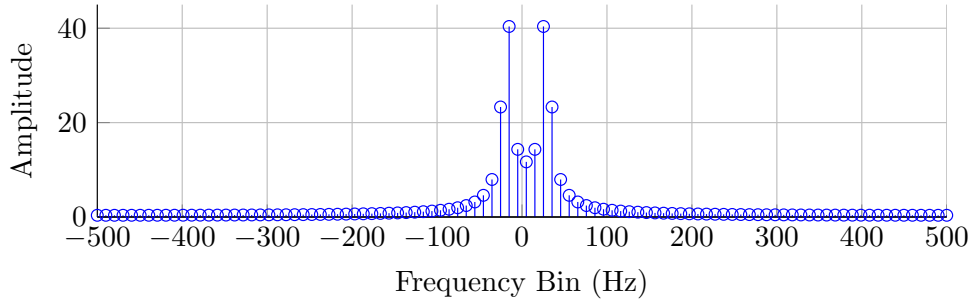


Figure 1.3: *DFT of a 100 sample 24Hz Sine Wave, with* $F_s = 1000Hz$

1.2 Properties of Power Spectral Density (PSO)

One definition of the PSD is from its DTFT:

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

Since it is complex, we can refer to it as

$$\begin{aligned} P(\omega) &= \lim_{N\to\infty} \mathbb{E}\left\{ \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-jn\omega} \sum_{k=0}^{N-1} x(k)^* e^{jk\omega} \right\} \\ &= \lim_{N\to\infty} \mathbb{E}\left\{ \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} x(n) \exp^{-jn\omega} x(k)^* e^{jk\omega} \right\} \end{aligned} \tag{2}$$

The autocorrelation sequence of a complex signal can be defined as

$$\begin{aligned} r_{xx}(n) &= \mathbb{E}\{x(k)x^*(k-n)\} \\ r_{xx}(k-n) &= \mathbb{E}\{x(k)x^*(n)\} \end{aligned} \tag{3}$$

Thus we can substitute this in to 2, to say

$$\lim_{N\to\infty} \sum_{n=0}^{N-1} \sum_{k=0}^{N-1} r_{xx}(k-n) e^{-j(n-k)\omega} \tag{4}$$

In [4], it is defined that a double summation can be converted to a single triangle summation, such that $\sum_{k=-M}^{M} \sum_{l=-M}^{M} g(k-l) = \sum_{\tau=-2M}^{2M} (2M+1-|\tau|) g(\tau)$. Substituting this in to our summation, we are left with

$$\lim_{N\to\infty} \sum_{\tau=-2N+2)}^{2N-2} (2N-1-|\tau|) r_{xx}(\tau) e^{-j\tau\omega} \tag{5}$$

From which we can see the clear relation to the DFT of the ACF: $P(\omega) = \sum_{k=-\infty}^{\infty} r(k) e^{-j\omega k}$

1.2.a Zero Padding

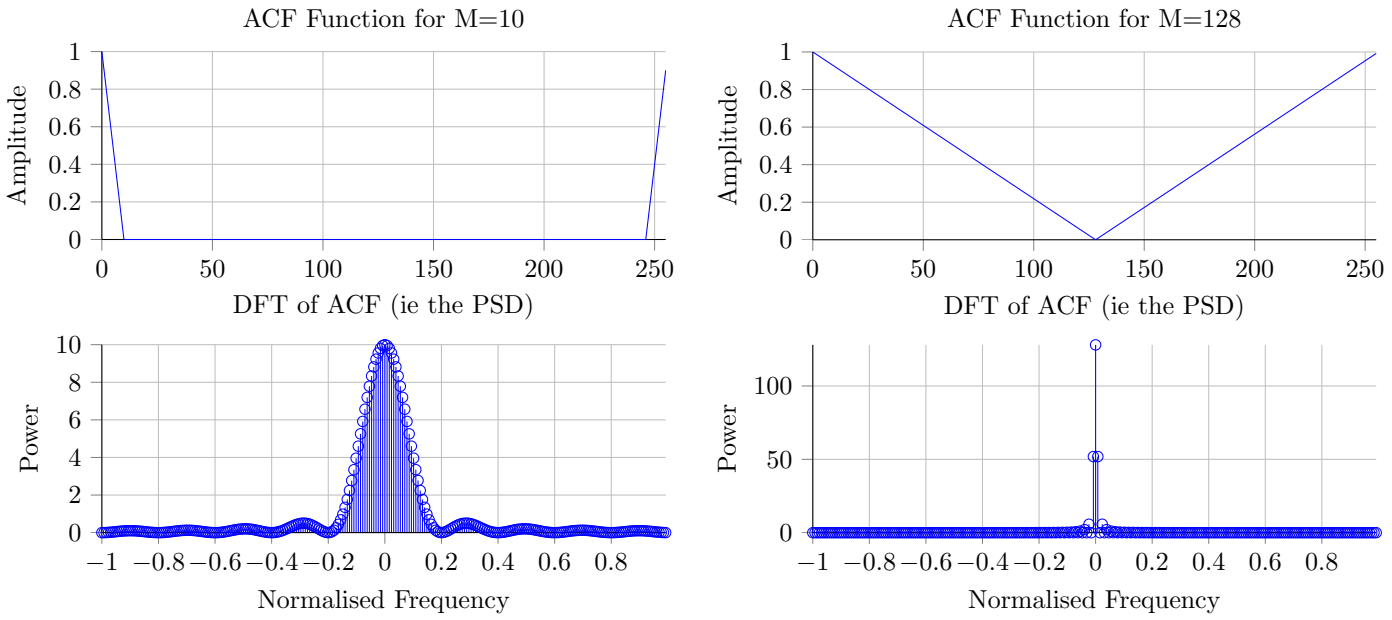
Figure **??** shows the Autocorrelation function for $M = 10$ and $M = 128$. Beneath the lag plot, we can see their DFTs. Since we are taking the DFT of the ACF, if this were in the time domain, it would be centred around $\tau = 0$. Since `MATLAB` does not have the concept of negative indices, we must adapt how we store our data. This results in the 'wrap around' which we observe in figure **??**. In the case of $M = 128$, no zero padding is used, since the signal is 268 samples long.

1.2.b The Real Component of the PSD

The DFTs in Figure **??** show the real components of the DFT of the ACF.

1.2.c Erroneous Spectral Values

By taking the incorrect wrap around, as observed for the modified ACF function, visible in Figure **??**, we observe negative power from the DFT of the ACF (which, as demonstrated earlier, represents the PSD). This is clearly incorrect (in this context, we cannot have negative power), and demonstrates the need to be careful when representing the ACF in a programming environment such as `MATLAB`. Effectively, we must wrap around the information which would have been stored in negative indices round to the last samples of the signal.



(a) *ACF for* $M = 10$ **(b)** *ACF for* $M = 128$

Figure 1.4: *Autocorrelation Functions and their DFTs*

1.2.d Shifting the DFT and Axis Scaling

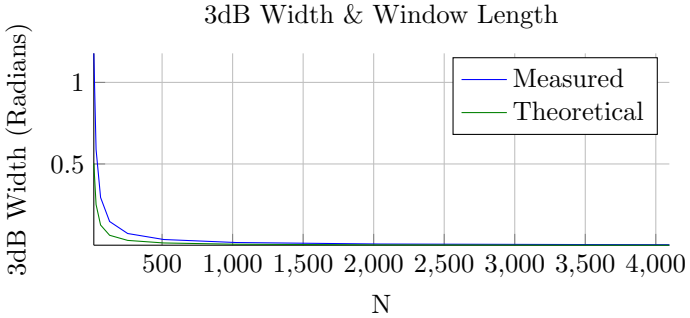
Through the figures so far, the custom function `limspace` (a play on the author's surname) has been used to set the axis scales. The key is to ensure that a sample is able to sit at 0, so the DFT is symettric about 0. The `MATLAB` function `linspace` returns a vector of linearly space points given an input, however it fails to take in to account the need for a value at 0. It does however work for odd length signals, since the point in the centre of the FT will naturally fall at 0. For even length signals, we generate a vector between 0 and the signal length - 1. This is then scaled by an 2 times arbitrary parameter (often 1, as has been the case in plots so far), and shifted down so that the central value of the vector represents 0.

By taking the IFFT (Inverse Fourier Transform) and using a shift, we can regenrate the true autocorrelation sequence they represented. This is shown in figure **??**. Figure **??** also shows odd length functions and how they are still correctly located on their axes.

1.3 Resolution and Leakage of Periodogram-based Methods

1.3.a Sidelobes of the Bartlett Window

Barlett Windows of varying lengths were generated from $N = 16$ to $N = 4096$, in powers of 2. Some of them have been shown in figure 1.8. Their frequency response both in linear terms and Decibels have been plotted. With these generated windows, the 3dB bandwith has been computed. The theoretical value of the 3dB bandwith (which represents when half of the full power is attenuated) is $0.89\frac{2\pi}{N}$. It has been plotted in figure 1.9, along with the measured results. We also look at the sidelobe height: the difference in gain between the main lobe and the first sidelobe. The theoretical value for a Bartlett Window is -27dB, and we can see the measured results in figure 1.10.



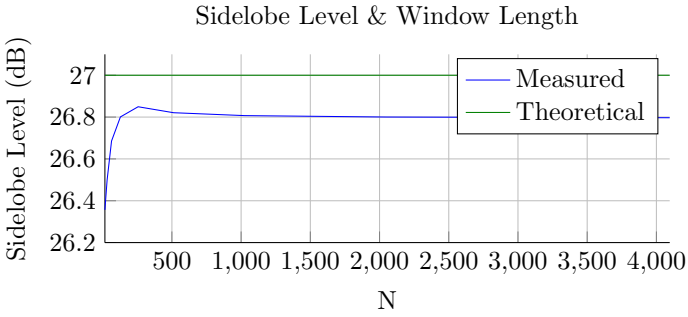
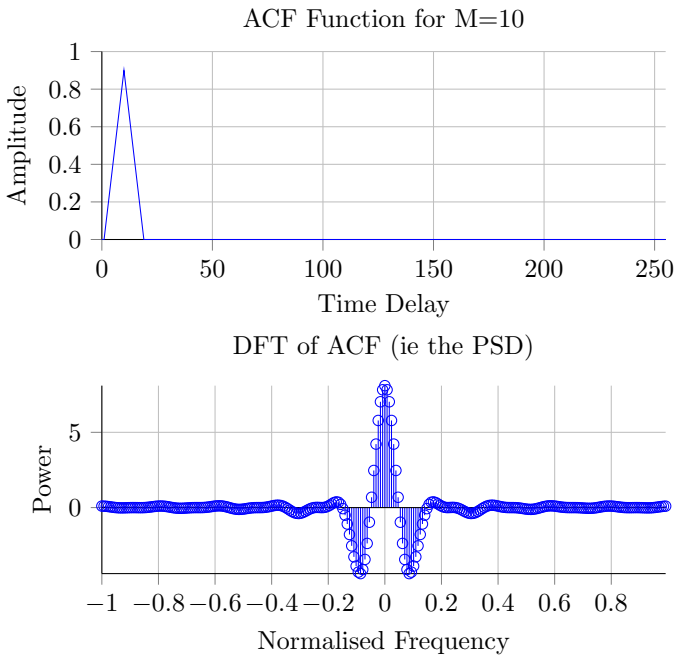


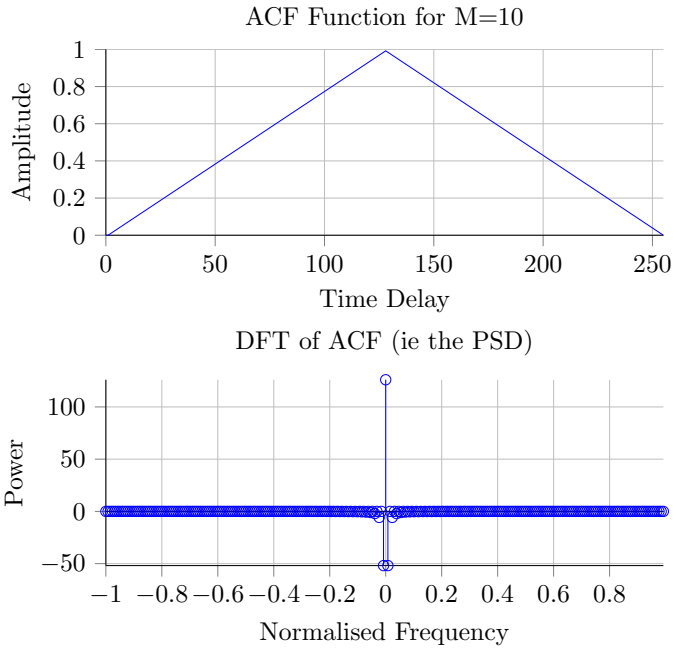
Figure 1.9: *3dB Bandwidth as a function of Window Length* **Figure 1.10:** *Sidelobe level as a function of Window Length*

1.3.b Resolution Threshold

Given the signal $x(n) = a_1 \sin(f_0 2\pi n + \phi_1) + a_2 \sin((f_0 + \frac{\alpha}{N} 2\pi n + \phi_2) + w(n)$ where $w \sim \mathcal{N}(0, \sigma^2)$, we pass this signal through a Barlett Window and observe the PSD. We fix all coefficients, remove the WGN as well as fixing the signal

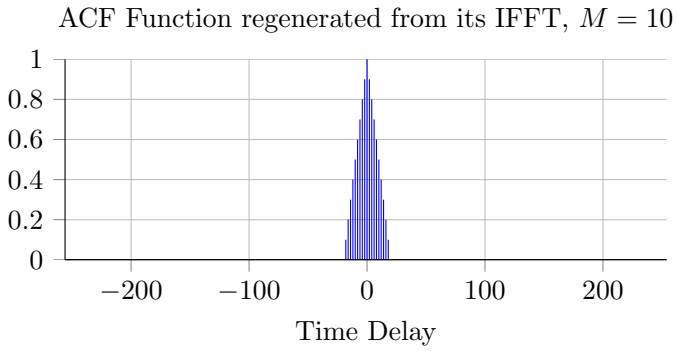


(a) *1 Tree*



(b) *2 Trees*

Figure 1.5: *Improperly shifted Autocorrelation Functions and their DFTs*



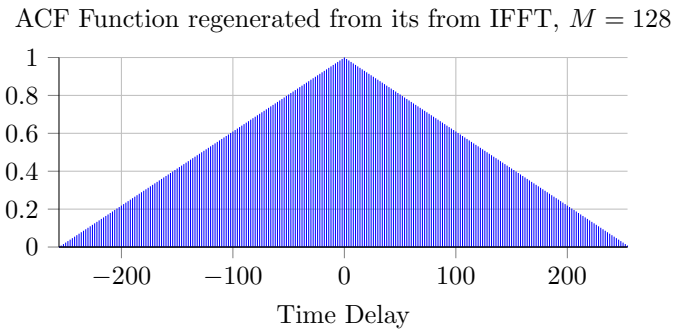


Figure 1.6: *Shifted IFFTs of the PSD*

length, only varying α. As α changes, the two peaks will become clear. This is shown for four different values of α in figure **??**. We can see at $\alpha = 0.65$, the two peaks start to appear. This is very clear by the time we reach $\alpha = 0.8$. α represents how close the two sinusoidal peaks are together. A larger α means the peaks are furhter away, and become easier to distinguish on the periodogram as two distinct sine waves.

1.3.c Resolution Threshold with a Hamming Window

We repeat the exercise conducted in 1.3.b, but this time using a Hamming Window instead of a Bartlett Window. In the interests of comparing the two functions, we keep α values consistent between them. It is clear from the plots in Figure **??** that the Hamming Window does not do as good a job as the Barlett Window in allowing us to identify to the two separate peaks. For example, at $\alpha - 0.65$, the Bartlett Window already shows signs of two peaks, but the Hamming Window does not.

1.3.d Leakage Effects on the Periodogram

Using the same signal as from 1.3.b, we fix α to 4 or 12, and vary a_2. We are spacing the sinusoids further apart than before, and analysing how different they can be in ampliude whilst still being distinguisable from each other. We can see in Figure 1.13 that by $a_2 = 0.01$ it is not distinguisable from the side lobes. Constrat taht with Figure 1.14 (where $\alpha = 12$), where for the same value of a_2 we can still make out the second sinusoid.

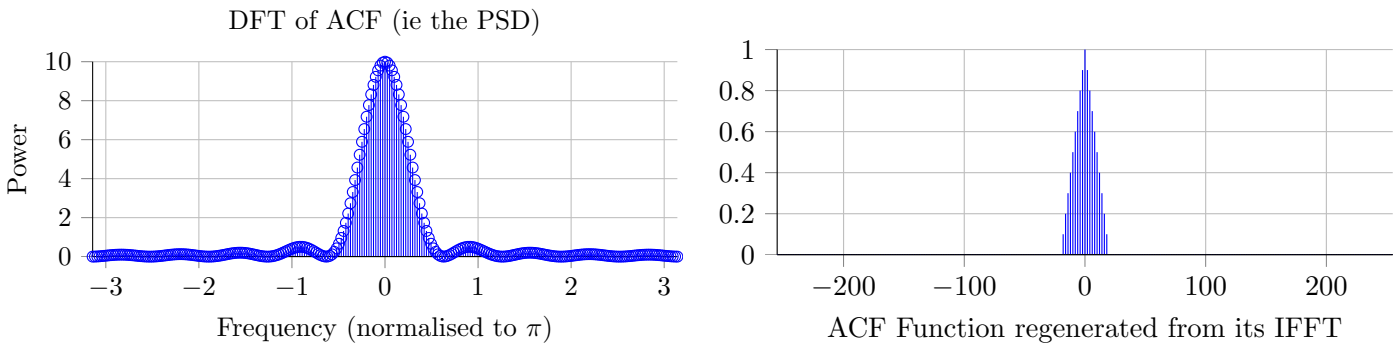


Figure 1.7: *Odd length DFT and IFFT functions, demonstrating the correct shift to reconstruct the ACF*

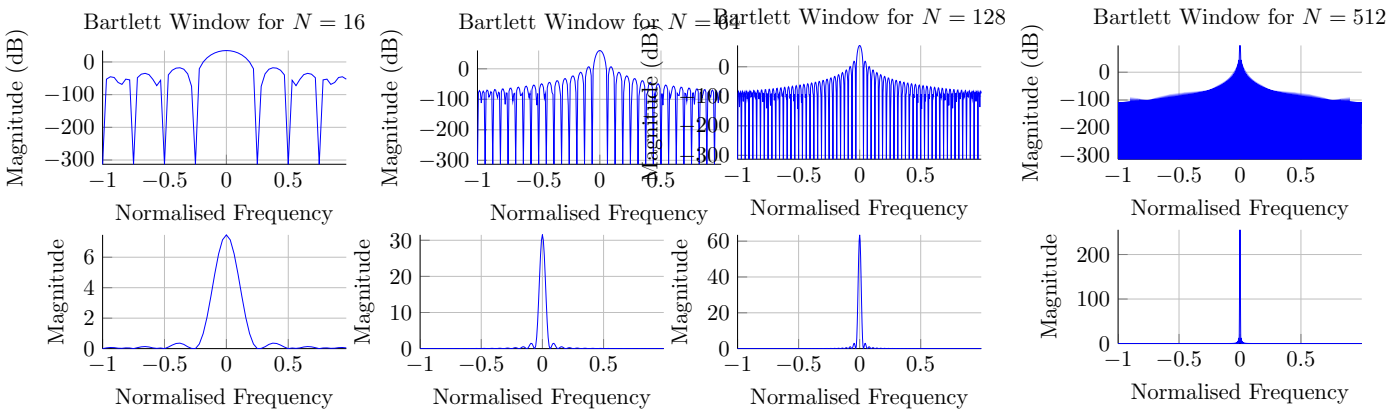


Figure 1.8: *Barlett Window Frequency Response: Linear Plots (lower) and Decibel Plots (higher)*

1.3.e The Effects of Alpha in relation to the Bartlett Window

As the value of α changes, it moves further and furhter away from the other sine wave. The further away it moves, the deeper in to the side lobes it goes. Once in the side lobes, the signal will be significantly attentuated and thus we will be unable to observe it. As we will see in the next point, we can use a Window with constant sidelobe height to help counteract this problem.

1.3.f Chebyshev and Blackman-Tukey Windows

The Chebyshev Window has a constant sidelobe height, meaning if the sinusoid is less liekly to get lost in a sidelobe. The Blackman-Tukey method is known sometimes as "periodogram smoothing" [4]. Effectively, it takes the Periodogram of the autocorrelation matric, aiming to create a more reliable measure of the PSD. Three different sidelobe heights were tested: 20dB, 50dB and 100dB. These are in Figures 1.15, 1.16 and 1.17, respectively. With a 20dB sidelobe, the second sinsoid is still visible in the Chebyshev window when $a_2 = 0.01$. Look at the 100dB sidelobes however, and there is still obviously the influence of the second sinusoid when $a_2 = 0.001$. This gives some idea of the power of choosing the correct window function. The Blackman-Tukey windows also show the presence of a second sinusoid. With data which is void of noise such as this. Its value may be more limited however. When we add noise in, it may prove more useful. In its current from, we are able to extract the information from the Chebyshev PSD directly.

1.4 Periodogram-based Methods Applied to Real-World Data

1.4.a The Sunspot Time Series

Presented with the Sunspot Time Series data, there are a number of preprocessing techniques we can apply to it before plotting the Periodogram. These are plotted in Figure 1.18. We observe that for the original data there is a high power at 0 frequency. This is the influence of the mean signal, so for other signals we mean centre the data. We can see that the detrended version (the black line) appears to suffer from less noise or jitter in the frequency domain. Taking the logarithm of data and mean centering it (the pink like) showed very interesting results. The line at points appears to jitter even less, for example at 0.35, the other lines suffer from a signficant drop, but it moves little by comparison.

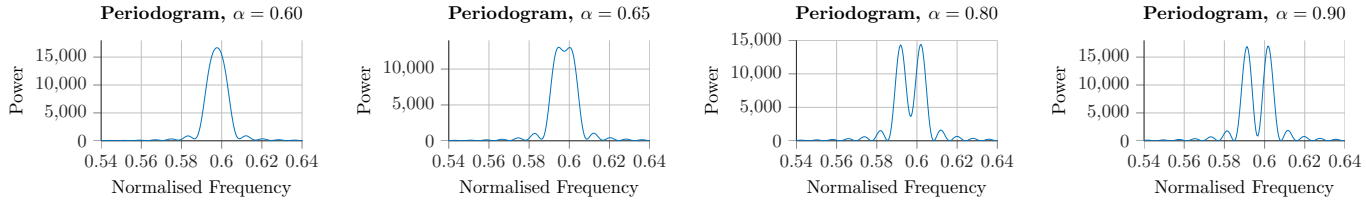


Figure 1.11: *Periodograms of Barlett Window with varying α of input signal*

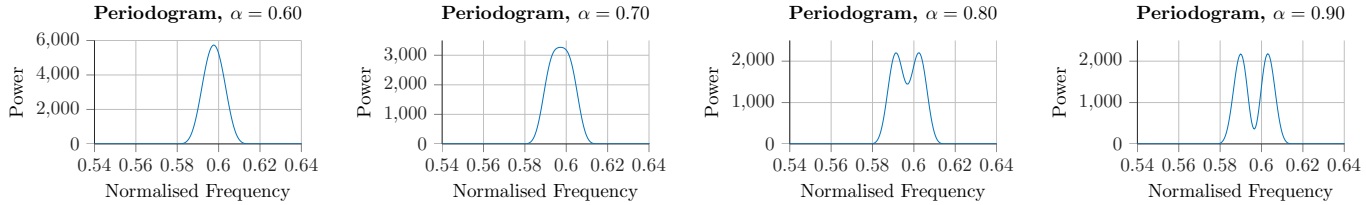


Figure 1.12: *Periodograms of Hamming Window with varying α of input signal*

1.4.b Brain Computer Interface (BCI)

We can see in figure 1.19 the periodogram taken for the entire signal. There is clearly a lot of noise, and thus analysing the periodogram itself to find the peak. Instead, we take the mean of windowed periodograms. Figures 1.20a, 1.20b and 1.20c show the mean periodograms for windows of 1, 5 and 10s respectively. They have also been run through Hanning and Bartlett windows, as an experiment to see if they imrpove the periodogram. The 1s periodogram is argubly too smooth and as such difficult to interpret any spike between $11Hz$ and $20Hz$ (although the lobe at $10Hz$ is noticable). The 10s periodogram may be a little too noisey, but the 5s mean does seem to represent a good balance between noise and still having obvious peaks. As such, we can estimate that there is a peak at approximately 13 Hz which represents the SSVEP.

In order to get a better quality periodogram, it may well be worth investigating the Welch method next time.

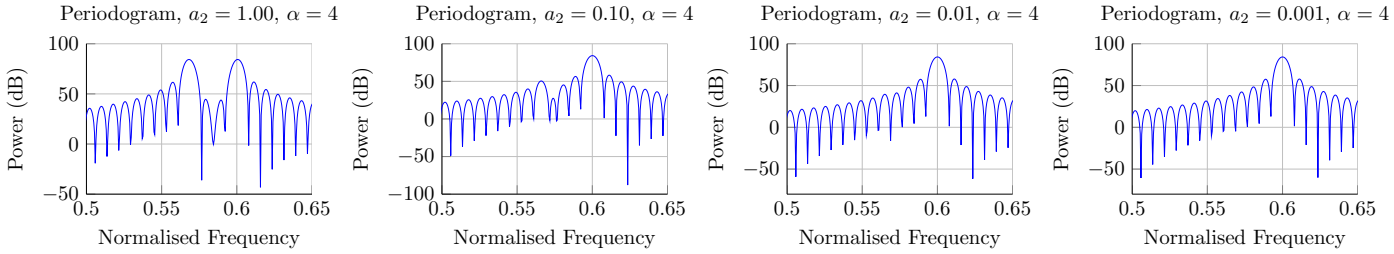


Figure 1.13: *Periodogram of varying amplitudes of a_2, with $\alpha = 4$*

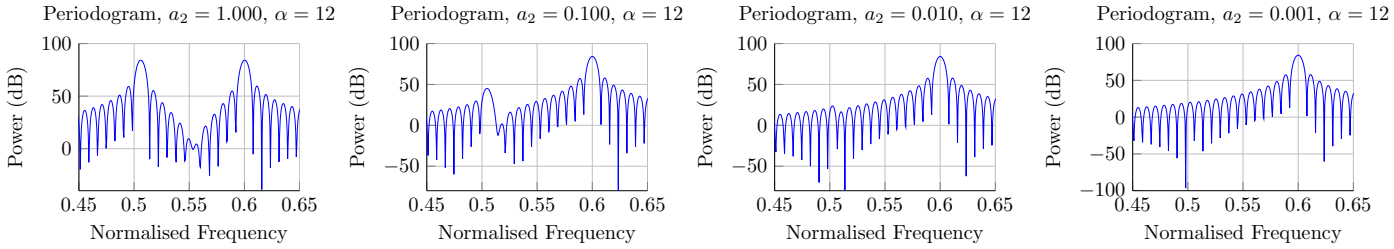


Figure 1.14: *Periodogram of varying amplitudes of a_2, with $\alpha = 12$*

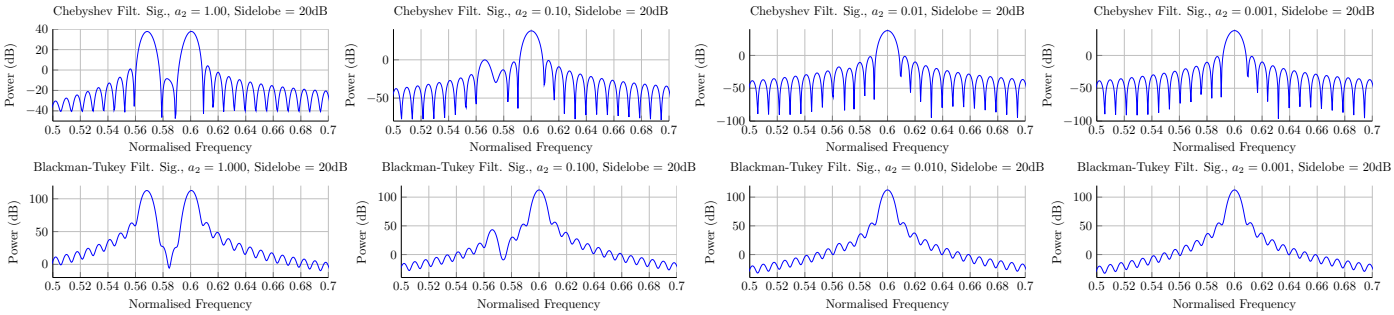


Figure 1.15: *Chebyshev and Blackman-Tukey with 20dB sidelobes*

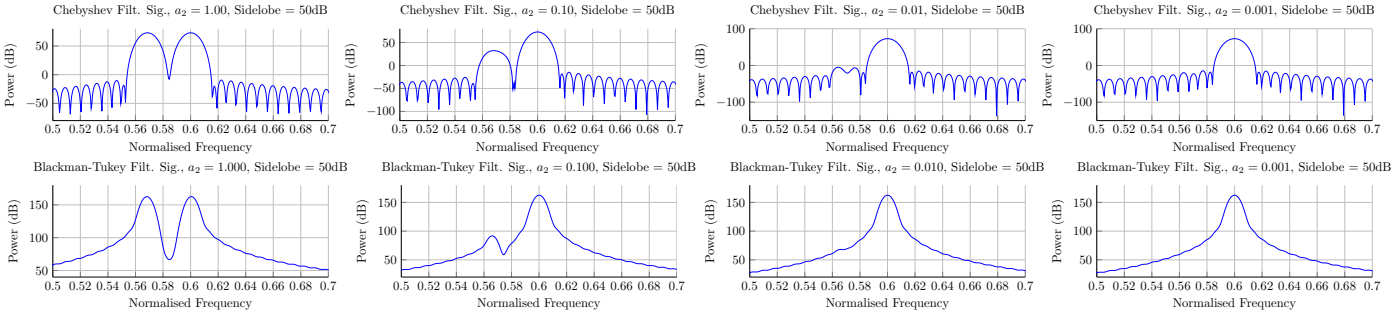


Figure 1.16: *Chebyshev and Blackman-Tukey with 50dB sidelobes*

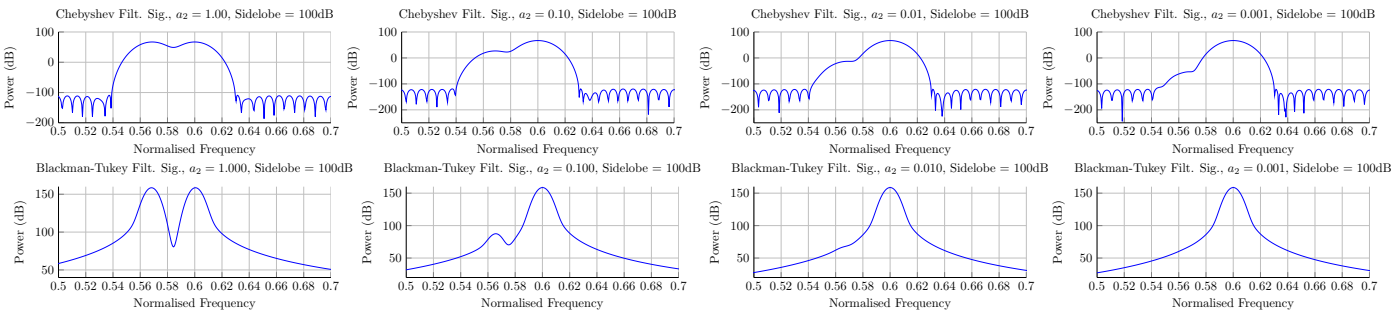


Figure 1.17: *Chebyshev and Blackman-Tukey with 100dB sidelobes*

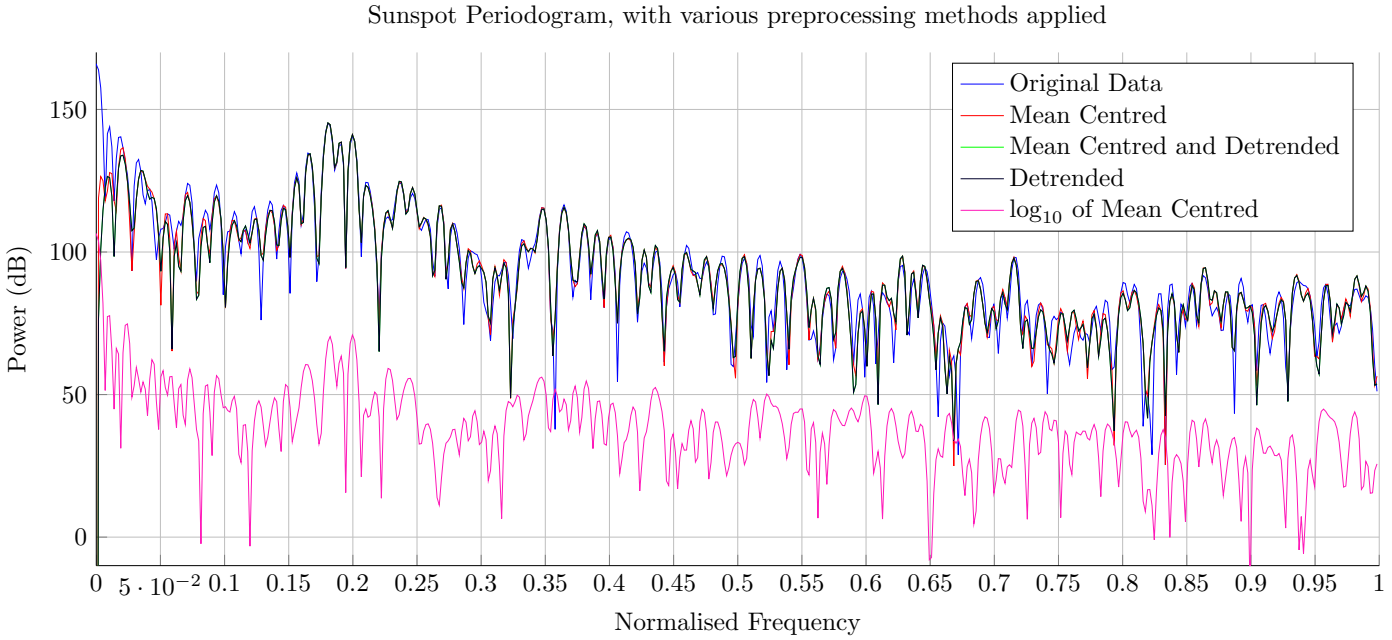


Figure 1.18: *Balanced and Unbalanced Complex Voltages*

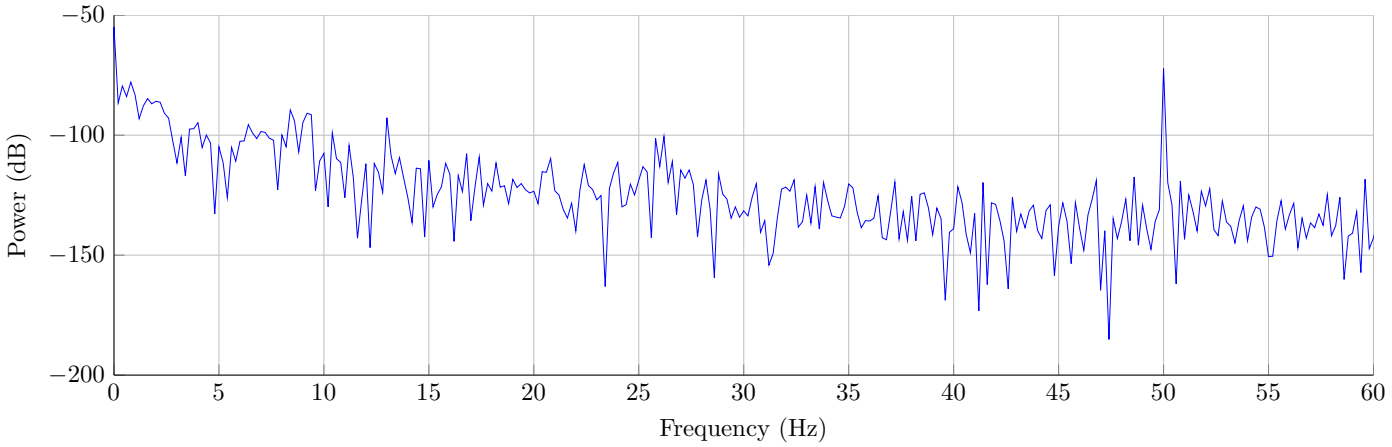
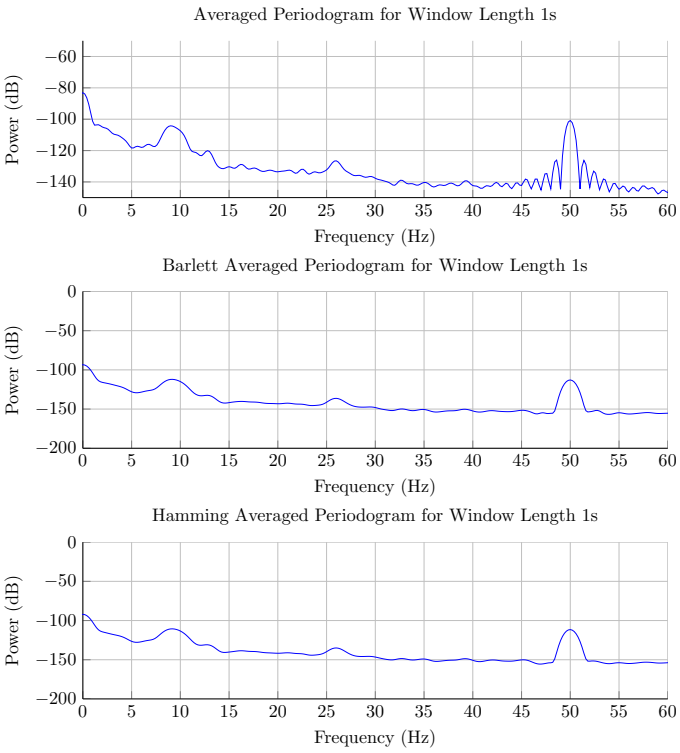
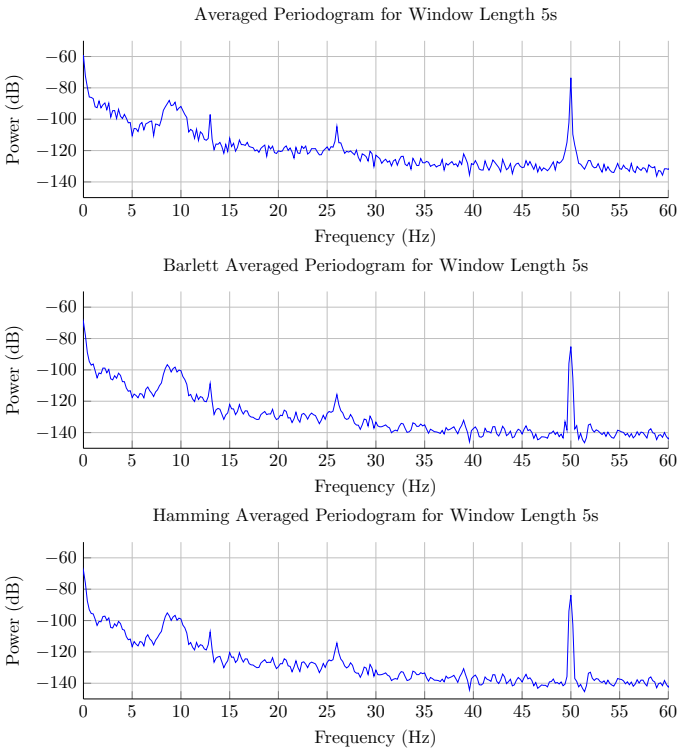


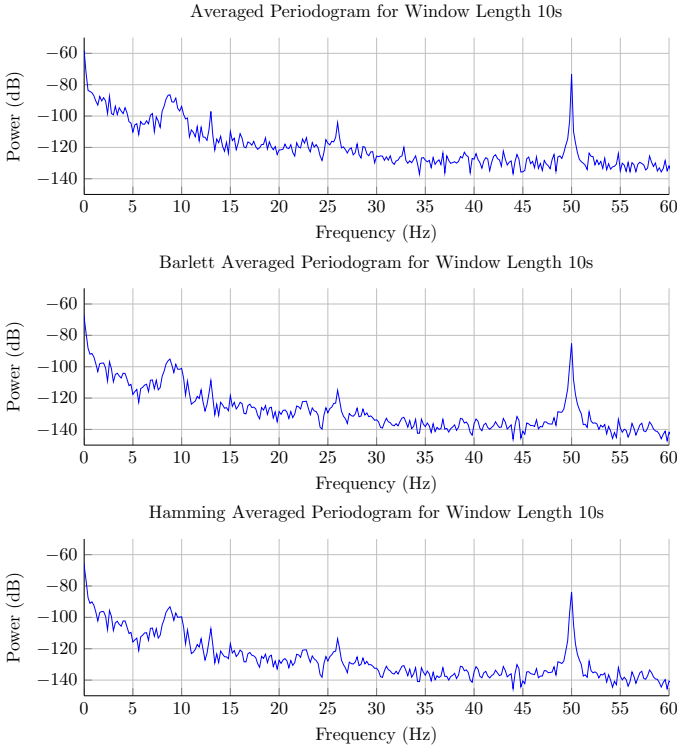
Figure 1.19: *Periodogram of entire signal length*



(a) *Mean Periodogram of 1s Windowed Periodograms*



(b) *Mean Periodogram of 5s Windowed Periodograms*



(c) *Mean Periodogram of 10s Windowed Periodograms*

2 Parametric and Line Spectra

2.1 Correlation Estimation

2.1.a Biased and Unbiased Autocorrelation, and the Correlogram

Figure 2.1 shows the ACF and Correlogram of Biased and Unbiased ACFs of WGN, Filtered WGN and a Sinusoid. We can see that that the biased ACF tends to get larger as the delay increases, due to the fewer samples which are interacting. The Biased ACF takes care of this, but has the downside that we now appear to be seeing significant amounts of negative power (which as discussed before, is not possible). The specral estimate for the sinusoids is exactly what we expect - a large peak at their operating frequnecies.

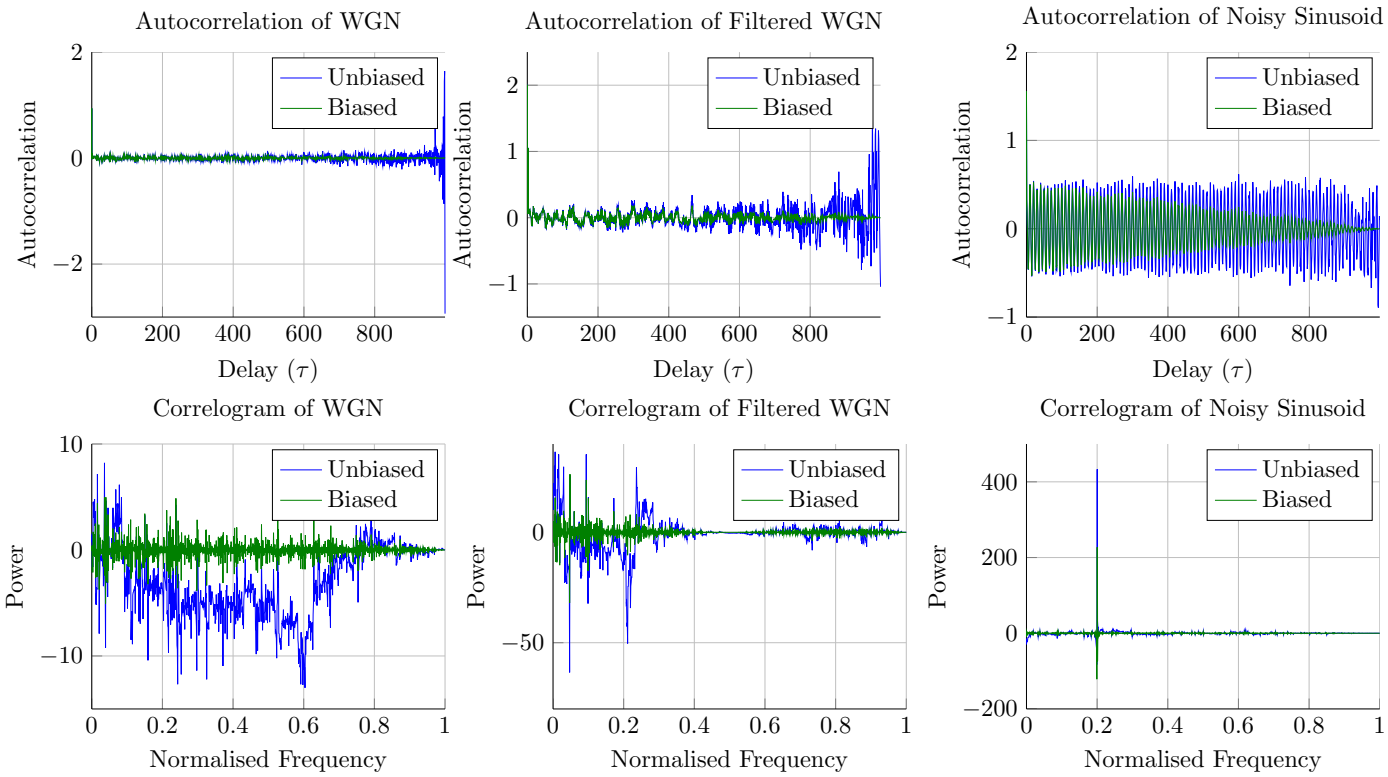


Figure 2.1: *Biased and Unbiased ACFs and Correlograms of WGN, Filtered WGN and a Sinusoids*

2.1.b PSD Generation through Ensembles

Figure 2.2 shows the ensemble iterations and mean for the WGN process. It is hard to see the real variation in the ensemble though, as they are all relatively small when compared to the peaks.

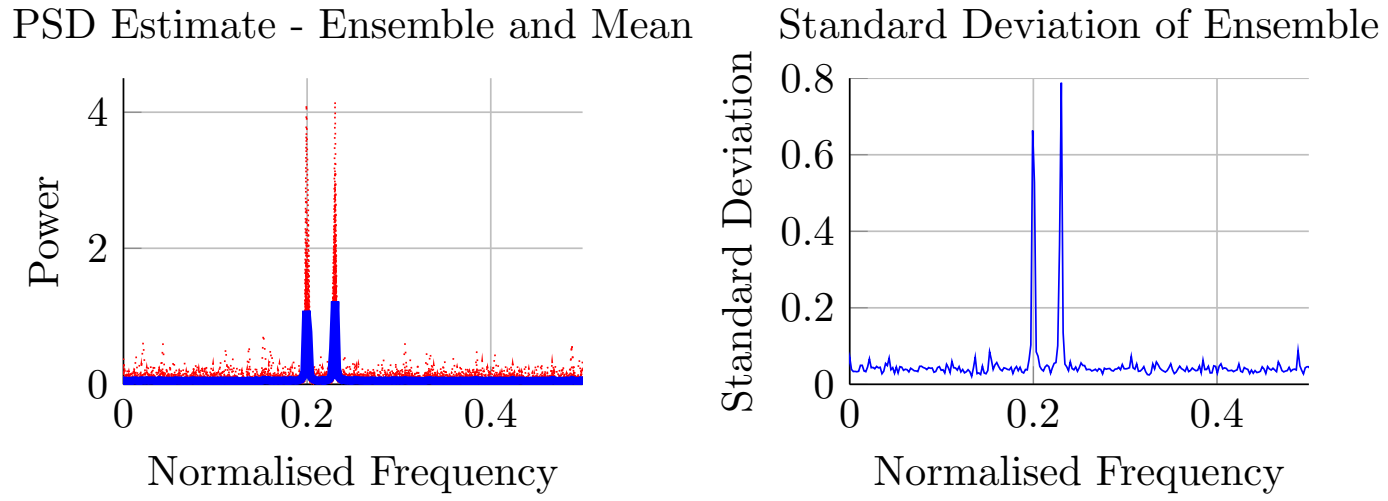


Figure 2.2: *PSD Estimate and Standard Deviation of Process*

2.1.c PSD Analysis in dB

Figure 2.3 shows the same plots as before, but this time in Decibels rather than a linear scale. It is now clearer to see the deviation of the ensemble than before, showing a vibrant contrast in the various iterations. The standard deviation is also clearer at getting the information across. Peaks are reduced, making it easier to see the trends within the data since the peaks are reduced in amplitude (when compared with the bulk of the data) - this is after all one of the main reasons for using a logarithmic scale.

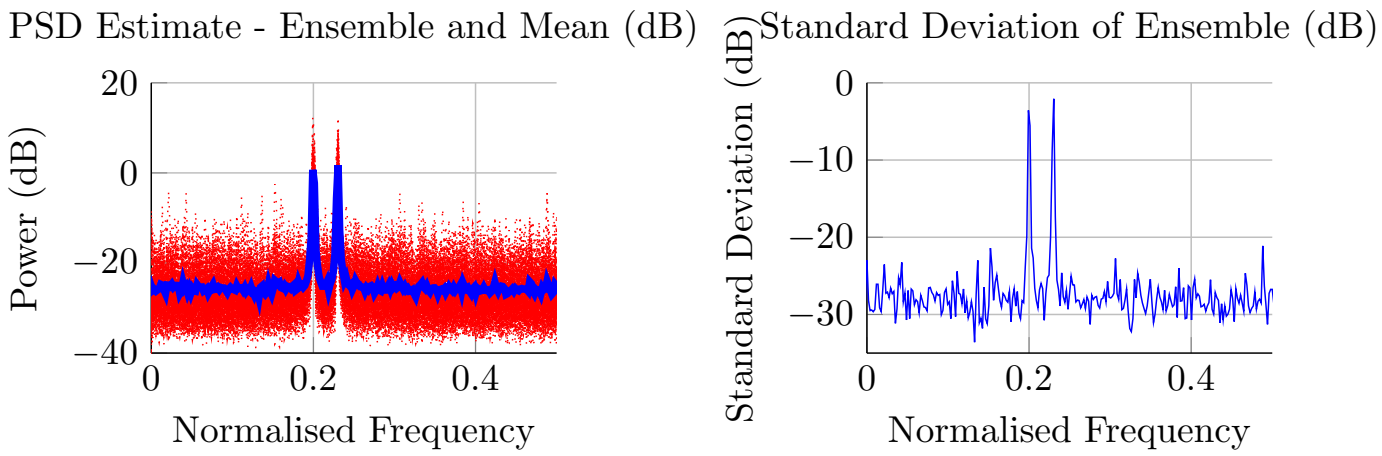


Figure 2.3: *PSD Estimate and Standard Deviation of Process - Decibels*

2.1.d Complex Exponential Signals

We have generated a complex signal as provided in code in the coursework handout. We can see that as we increase the signal length (but the size of the DFT is fixed at $N = 128$), as shown in figure 2.4, the two sinusoids become more noticable. In the signal's 'default' parameters as shipped, we are unable to make out the two different sinusoids. In Figure 2.5 we vary the frequency of one of the sinusoids. It becomes clear that by increasing the gap between sine waves, the periodogram starts to show the correct spectra.

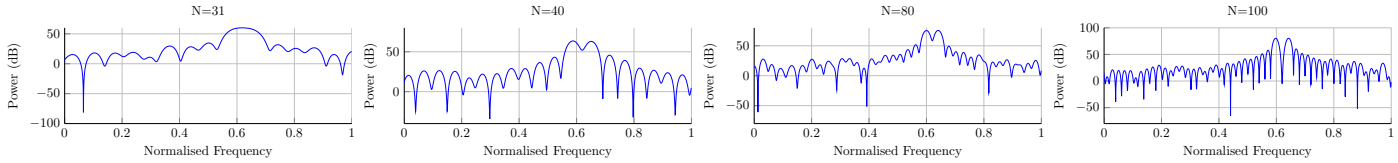


Figure 2.4: *Varying Signal Length N*

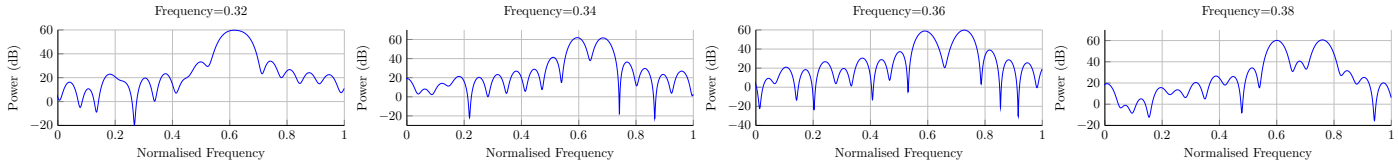


Figure 2.5: *Varying Frequency for one of the sinusoids*

2.1.e MUSIC Method

Figure 2.6 shows the output of the MUSIC method. It is clearly able to clearly mark the two different spectra in a very different fashion to those seen from PSD estimates.

2.2 Spectrum of Autoregressive Processes

2.2.a Shortcomings of the Unbiased ACF

As discussed in [4], the unbiased ACF may not always be invertible. In order to use modern spectrum estimation techniques such as this section, the autocorrelation matrix must be invterible. If it is not, we are unable to continue using the Yule Walker equations to compute weights for the estimated model.

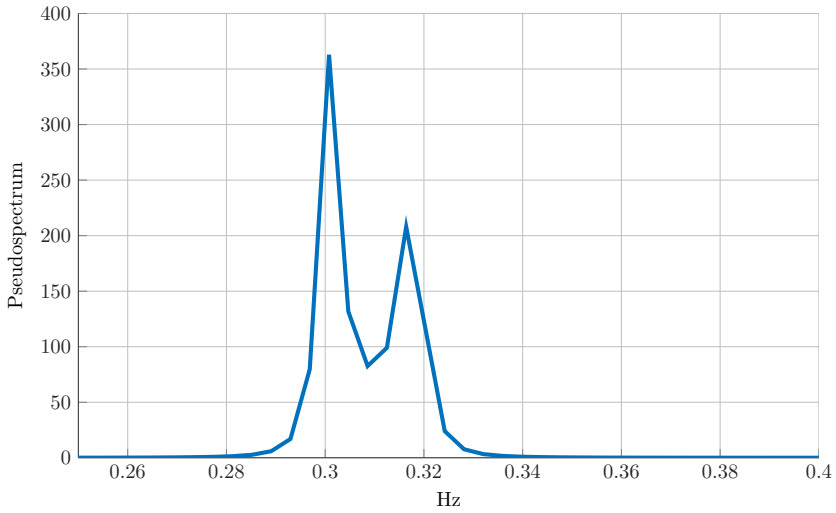


Figure 2.6: *MUSIC PSD Estimation*

2.2.b Modelling an AR Process

Given the equation $x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n)$ where $w \sim \mathcal{N}(0,1)$. We are also adviced to discard the first 500 samples as the filter initiailises itself. Given a range of potential model orders to estimate the signal from, we find that an AR(5) process is liekly the best fit, as determined by visual inspection. There appears to be little value increasing the model order significantly - as the peaks come more defined, but overall they add little more.

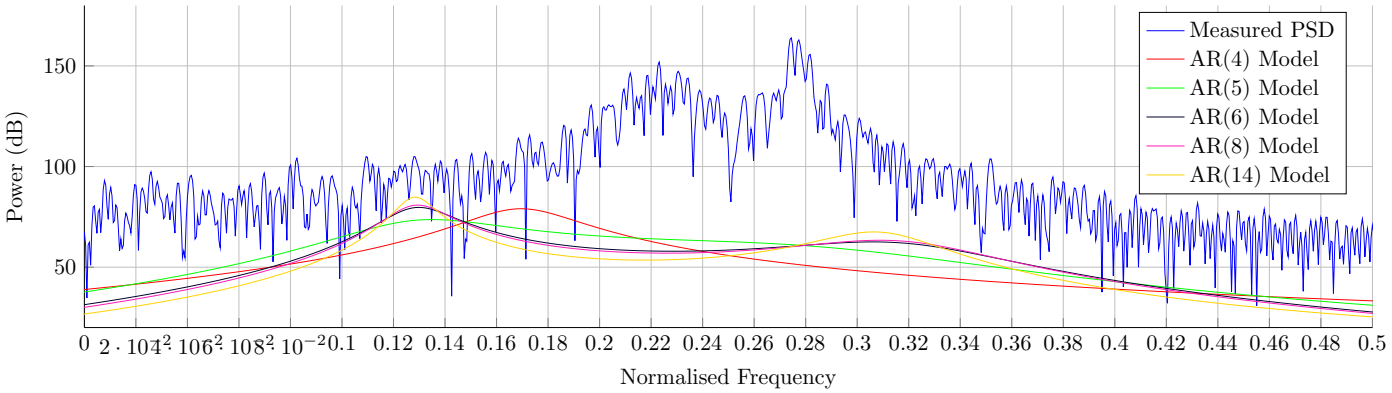


Figure 2.7: *PSD of Signal, and Estimated Models for varying orders*

2.2.c Increasing data length when modelling

Figure 2.8 shows the same plot as before, but this time with 10,000 samples rather than 1,000 as before. We can see clear under modelling with an attempted AR(2) process - it shows only one peak rather than the two in the data. Once again, there appears to be little value from over modelling since the extra weights appear to do very little to improve the estimate of the system. As before, the best fit does to be from the AR(5) estimate.

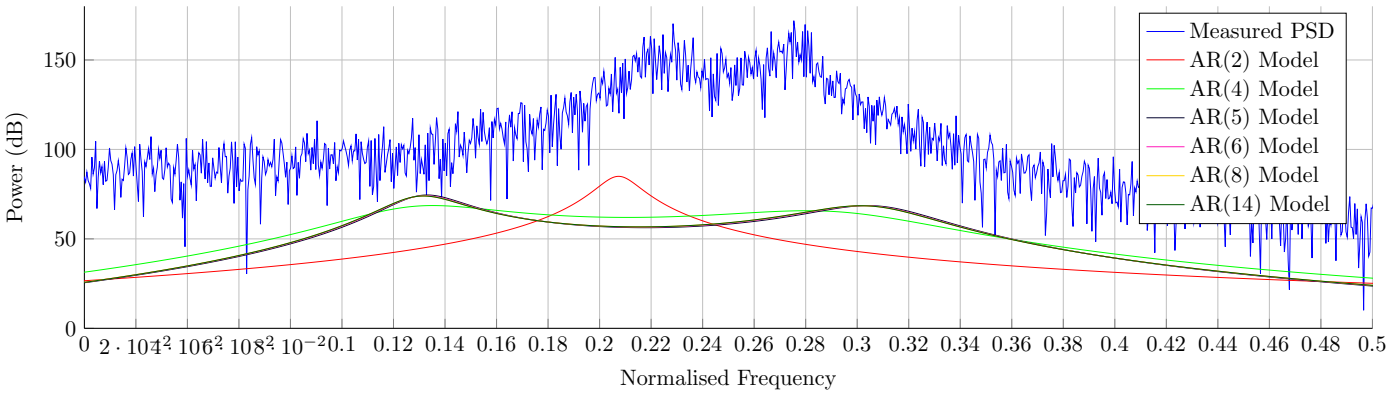


Figure 2.8: *PSD of Signal, and Estimated Models for varying orders - with 10,000 samples*

2.3 Time-Frequency Estimation

2.3.a Experimenting with the Spectrogram

We are introduced to the concept of the spectrogram, which allows us to view the frequency spectrum across the time domain. For example, a the frequency spectra of a voice signal will change significantly over time, and the spectrogram allows us to view this change without looking at a different PSD for each period in time.

There are various parameters which go in to the spectrogram. The `chirp` function from `MATLAB` was used to generate the signal used across these tests. Figure 2.9 shows differing window sizes - the window taken for each Short Time Fourier Transform. The window also allows a smooth FFT (so as to avoid the problems of using a rect gate)- we will look at some different Windows (for example, Bartlett). There is also a proportion of overlap between each segment, so that the spectrogram appears smoother between each step. The legnth of the FFT also has an impact, as it will determine the resolution of the spectrogram at each step.

Figure 2.9 shows different window sizes. To ensure a fair test, we keep the FFT of the same length. The two middle window lengths, 128 and 256 appear the smoothest. A shorter window length results in noticably large chunks in the time domain. Conversely, a long window suffers from having too large a bandwith and thus having poor resolution in the freqency domain (noice how wide the chirp appears to be in comparison to the others).

Figure 2.10 shows differing FFT lengths. Clearly, the larger the FFT the better the resolution. There is a trade off to be made between computational complexity and FFT length, however.

Figure 2.11 has 4 different percentages of overlap. This is allows a smooth transition between time steps, We can see that the overlap of 80% appears the smoothest out of all overlaps. Of course, with such a large overlap, this means that more windows and STFTs will need to be computed.

Lastly, we look at differnet window functions in figure 2.12. Judging by the spectrogram, the Hanning window appears to have performed the best given the current set up.

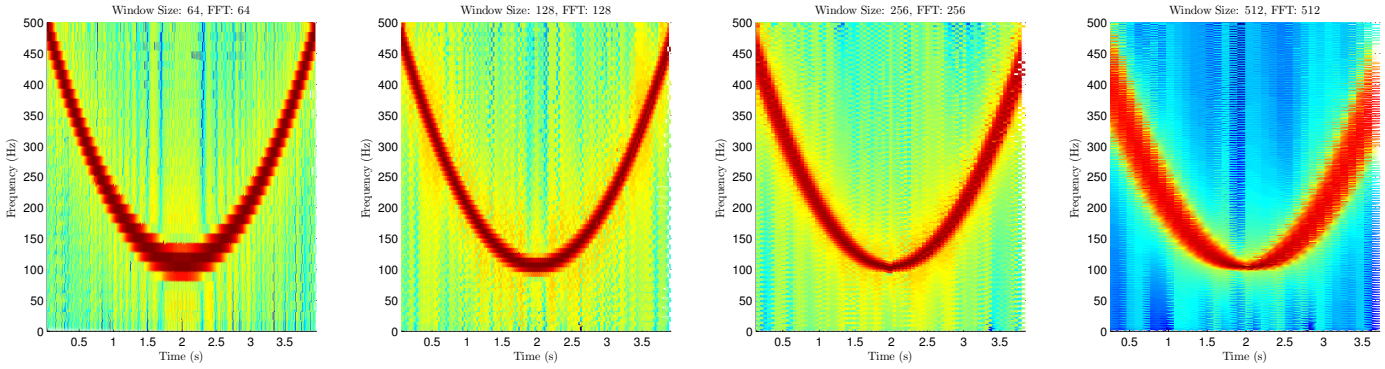


Figure 2.9: *Differing Window Sizes*

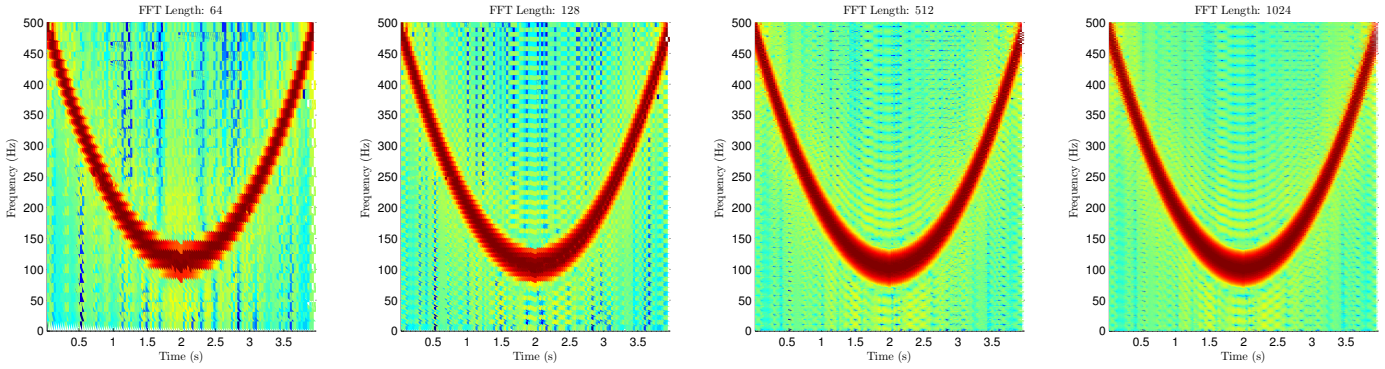


Figure 2.10: *Differing FFT lengths*

2.3.b Spectrogram of Real-World EEG Data

Having understood the basics of how to create a spectrogram, we now look at creating one for some real-world data. In this case, an EEG signal. In order to achieve good spectral resolution, which pick an FFT length of 4096, nearly four times the sampling frequency ($1200Hz$). Based on the performance in tests conducted, a window length of $\sim 75\%$ was selected (3000 samples). The Hanning window performed best in the last spectrogram test, so was also used here.

Figure 2.13 shows the final spectrogram. We appear to have good frequency and time performance. We can see very clearly the 50Hz mains interference, as well as activitiy between 10 and $20Hz$. We also have the spectral resolution to observe the $13Hz$ line, which otherwise may have been blurred in with the high amplitude noise nearer $10Hz$.

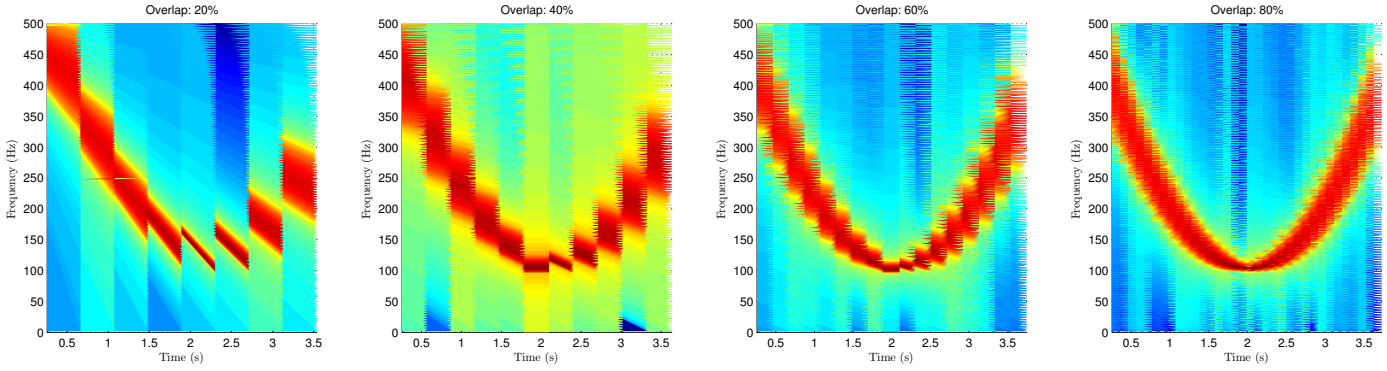


Figure 2.11: *Differing Window overlaps*

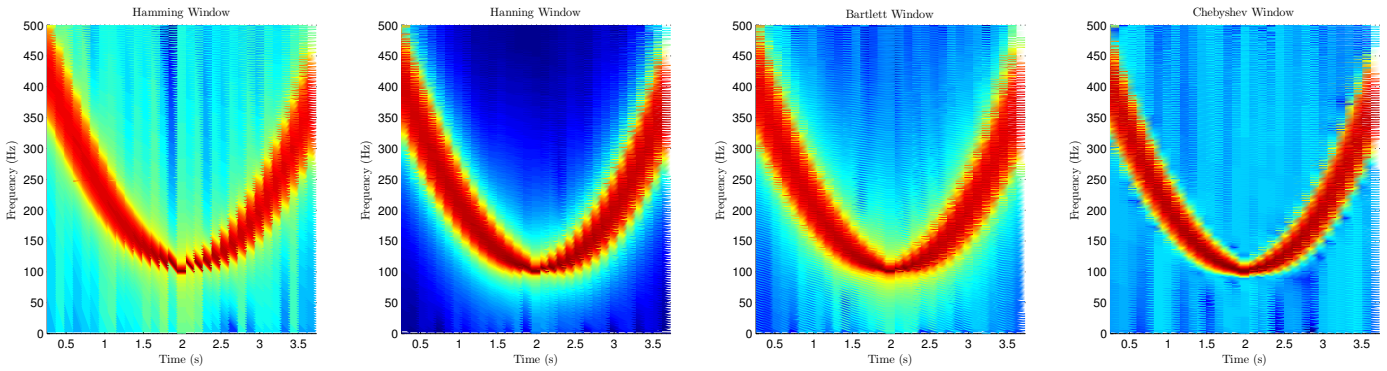


Figure 2.12: *Different window functions*

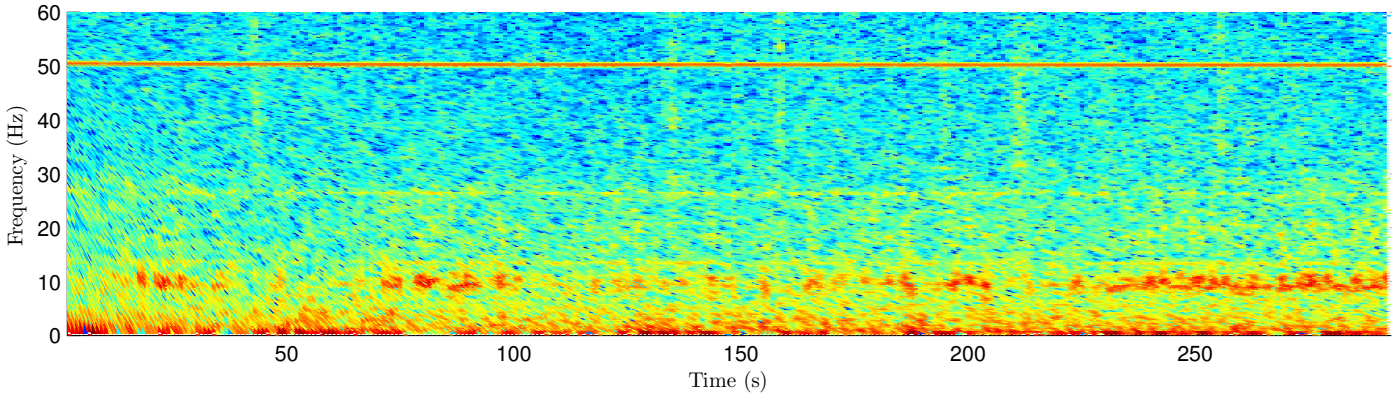


Figure 2.13: *Spectrogram of EEG data*

3 Adaptive Signal Processing

3.1 The Least Mean Squares (LMS) Algorithm

3.1.a Correlation Matrix

The autocorrelation matrix is been defined as $R \equiv E\left[\mathbf{x}[n]\mathbf{x}^T[n]\right]$, where $\mathbf{x}(n) = [x(n-1), x(n-2)]^T$

$$R = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) \\ R_{xx}(1) & R_{xx}(0) \end{bmatrix} = E\left\{ \begin{bmatrix} x^2[n-1] & x[n-1]x[n-2] \\ x[n-1]x[n-2] & x^2[n-2] \end{bmatrix} \right\} = \begin{bmatrix} E\{x^2[n-1]\} & E\{x[n-1]x[n-2]\} \\ E\{x[n-1]x[n-2]\} & E\{x^2[n-2]\} \end{bmatrix} \tag{6}$$

We note that $x^2[n-1] = x^2[n-2]$ since this simply represents a shift in the time domain but will not change the expected value.

We can square $x[n-1]$ to give us the expected value of the diagonals:

$$E[x^2[n-1]] = E[a_1^2x^2[n-2] + a_2^2x^2[n-3] + 2a_1a_2x[n-1]x[n-2] + \eta[n-1](a_1x[n-2] + a_2x[n-3]) + \eta^2[n-1]] \tag{7a}$$

$$E[x^2[n-1]] = E[a_1^2x^2[n-2]] + E[a_2^2x^2[n-3]] + E[2a_1a_2x[n-1]x[n-2]] + \sigma^2 \tag{7b}$$

$$R_{xx}(0) = E[x^2[n-1]] = a_1^2R_{xx}(0) + a_2^2R_{xx}(0) + 2a_1a_2R_{xx}(1) + \sigma^2 \tag{7c}$$

We get to the last line using the equality mentioned above, and we can see the make up of $R_{xx}(1)$ in equation 6. We can conduct a similar process for $R_{xx}(1)$:

$$E[x[n-1]x[n-2]] = E[a_1x^2[n-2] + a_2x[n-3]x[n-2] + \eta[n-1]x[n-2]] \tag{8a}$$

$$R_{xx}(1) = E[x[n-1]x[n-2]] = a_1R_{xx}(0) + a_2R_{xx}(1) + 0 \tag{8b}$$

We then solve these two equations for $R_{xx}(0)$ and $R_{xx}(1)$, to determine that the autocorrelation matrix is

$$R = \begin{bmatrix} \frac{25}{27} & \frac{25}{54} \\ \frac{25}{54} & \frac{25}{27} \end{bmatrix}$$

In order for the filter to converge to the correct parameters, we must satisfy the bounds $0 < \mu < \frac{2}{\lambda_{max}}$. In this case, our eigenvalues are $\frac{25}{18}$ and $\frac{25}{54}$. Thus we know that $0 < \mu < \frac{36}{25} (= 1.44)$ for the LMS to converge in the mean.

3.1.b Implemented LMS Filter

100 iterations of the AR Process $x[n] = 0.1x[n-1] + 0.8x[n-2] + \eta[n]$ have been generated, with 1000 samples per iteration. Figure 3.1 shows one trial of this filter, whist figure 3.2 shows the mean error taken across 100 iterations. We can see that 100 iterations shows a much clearer trend in the error decreasing and becoming steady at around 300 iterations in. The steady state is around -6dB of prediction error. The Mean Squared Error is defined as $\sigma_\eta^2 + \text{EMSE}$ (discussed further in section 3.1.c), this term is dominated by the power of the noise, σ_η^2. We know for this system that $N \sim \mathcal{N}(0, 0.25)$, and if we convert that power to dB, it is -6dB.

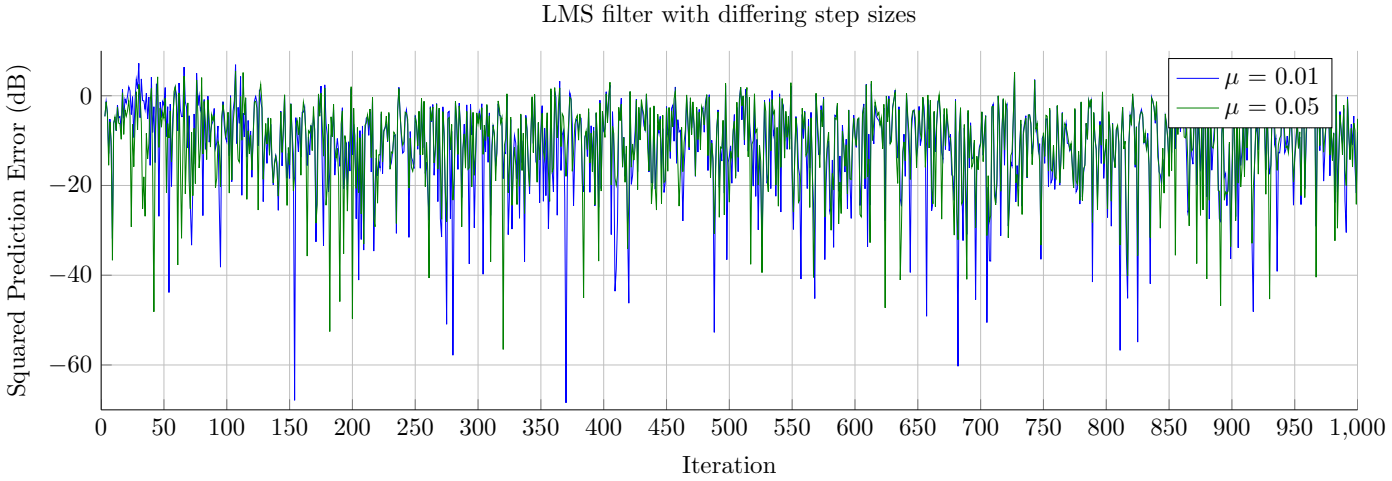


Figure 3.1: *LMS filter error (dB) for the given AR process*

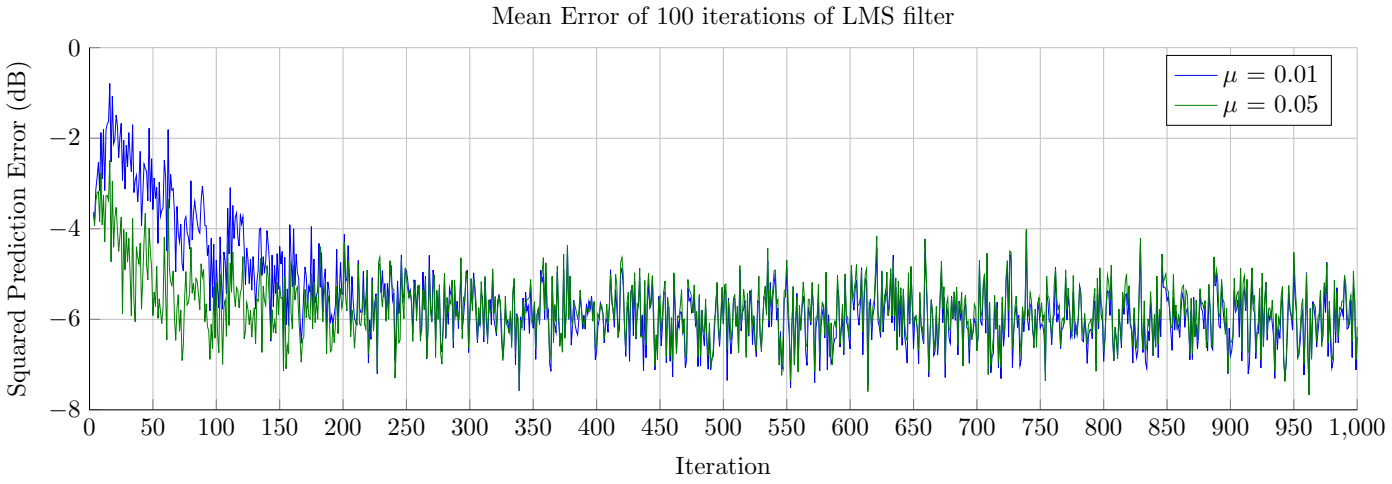


Figure 3.2: *LMS filter error (dB) for the given AR process, averaged over 100 independent trials*

3.1.c Misadjustment

Misadjustment is defined as $\mathcal{M} = \frac{\text{EMSE}}{\sigma_\eta^2}$, where the total mean squared error MSE $= \lim_{n\to\infty} \mathbb{E}\{e^2(n)\} = \sigma_\eta^2 + \text{EMSE}$. Thus we can compute the measured misadjustment from the known mean squared error, and variance of the White Gaussian Noise. We also know that we can approximate $\mathcal{M} \approx \frac{\mu}{2}\text{Tr}\{\mathbf{R}\}$ (for small μ) which allows us to have an idea of its theoretical values.

Based on the data obtained in section 3.1.b, we have taken the mean error squared of the last two hundred samples (when it is known to be in a steady state) across the 100 iterations of the LMS filter, and computed the misadjustment. Table 1 shows the theoretical and measured misadjustment values. The results are close to the theoretical values (and certainly all of the same order), although running the test multiple times does show that the misadjustment varies between each set of 100 trials. It is worth noting that the error (difference away from the theoretical value) at $\mu = 0.01$ seems to be greater than the error for $\mu = 0.05$.

μ	Theoretical $\mathcal{M}$	Measured $\mathcal{M}$
0.01	0.0093	0.0028
0.05	0.0463	0.0456

Table 1: *Theoretical and Actual Misadjustment values measured for different step sizes*

3.1.d Steady State Coefficient Values

By taking the mean of the two hundred last values (when we know the filter is in steady state), and then taking the mean across 100 iterations, we can get estimations of the coefficients a_1 and a_2. The results are shown in table 2. We can see that the coefficient estimates for $\mu = 0.01$ is nearer to the true values than for $\mu = 0.05$. We know that there will always be some error, due to the power of the WGN signal (the σ^2 term in the Mean Squared Error equation). We also know from section 3.1.c that the misadjustment for $\mu = 0.05$ is larger than for $\mu = 0.01$, hence it is not surprising that it also less close to the known values than $\mu = 0.01$. However, we know from Figure 3.2 that a larger step size results in faster convergence to the correct value, so it is a case of a trade off between a small steady state error, and fast convergence.

μ	True Values	0.05	0.01
a_1	0.1	0.0689	0.0943
a_2	0.8	0.7176	0.7691

Table 2: *Theoretical and Actual Coefficient values measured for different step sizes*

3.1.e Leaky LMS Derivation

In order to derive the leaky LMS equation, we differentiate the given cost function, using the known result of $\nabla(e^2(n))|_{w_n}$, and that $\nabla(\|\mathbf{x}(n)\|_2^2)|_{x_n} = 2\mathbf{x}$ [1] :

$$J(n) = \frac{1}{2}(e^2(n) + \gamma\|\mathbf{w}(n)\|_2^2) \tag{9a}$$

$$\nabla J(n)|_{w_n} = \frac{1}{2}(\nabla(e^2(n))|_{w_n} + \gamma\nabla(\|\mathbf{w}(n)\|_2^2)|_{w_n}) \tag{9b}$$

$$\nabla J(n)|_{w_n} = -e(n)\mathbf{x}(n) + \gamma w(n) \tag{9c}$$

We then substitute this in to the update function:

$$\mathbf{w}(n+1) = \mathbf{w}[n] + \mu(-\nabla J(n)|_{w_n}) \tag{10a}$$

$$\mathbf{w}(n+1) = \mathbf{w}[n] + \mu e(n)\mathbf{x}(n) - \mu\gamma\mathbf{w}(n) \tag{10b}$$

$$\mathbf{w}(n+1) = (1 - \mu\gamma)\mathbf{w}[n] + \mu e(n)\mathbf{x}(n) \tag{10c}$$

3.1.f Leaky LMS Results

Table 3 shows estimated parameters for a_1 and a_2 with varying μ (the same values used in section 3.1.b) and γ values. We can see that as the leakage parameter γ increases, the coefficient estimates get progressively less accurate. [3] defines $\lim_{n\to\infty}\mathbb{E}[\mathbf{w}_n] = (\mathbf{R} + \gamma\mathbf{I})^{-1}\mathbf{p}$. The new term added is the $\gamma\mathbf{I}$ term, which effectively introduces noise in to the input matrix $\mathbf{x}$. Its purpose is to allow the input matrix to have non-zero eigenvalues, as in some cases this can happen, meaning coefficients do not converge. Since the extra term is effectively adding white noise, this explains why the coefficient estimates get progressively worse as γ increases. Since the purpose of Leaky LMS is to help coefficients converge if they otherwise cannot, care should be taken when to add the γ term in, and when it is used to keep it as small as possible, so as to ensure the most accurate results are obtained.

	$\mu = 0.01$	$\mu = 0.05$
$\gamma = 0$	$a_1 = 0.0939$, $a_2 = 0.7694$	$a_1 = 0.0695$, $a_2 = 0.7162$
$\gamma = 0.2$	$a_1 = 0.1320$, $a_2 = 0.6078$	$a_1 = 0.1043$, $a_2 = 0.5463$
$\gamma = 0.5$	$a_1 = 0.1412$, $a_2 = 0.4682$	$a_1 = 0.1105$, $a_2 = 0.4110$
$\gamma = 0.8$	$a_1 = 0.1349$, $a_2 = 0.3836$	$a_1 = 0.1050$, $a_2 = 0.3333$

Table 3: *Estimated AR Coefficients for different μ and γ*

3.2 Adaptive Step Sizes

3.2.a Implemented GASS Algorithms

Gradient Adaptive Step-Size Algorithms are designed to change the step size as the algorithm progresses. They all rely on optimising gradient descent so as to iterate quickly down a steep gradient, and then remain very stable when in steady state At the start, a large step size is needed in order for the algorithm to converge quickly, but during steady state a small step size helps to reduce the steady state error (as we have observed in previous parts of this coursework).

The LMS algorithm remains the same in structure, but μ now becomes $\mu(n)$, thus becoming $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu(n)e(n)\mathbf{x}(n)$. The algorithm to update the weight is defined as $\mu(n+1) = \mu(n) + \rho e(n)\mathbf{x}^T(n)\boldsymbol{\psi}(n)$. $\boldsymbol{\psi}(n)$ is defined by three different GASS algorithms, listed below:

Matthews & Xie $\boldsymbol{\psi}(n) = e(n-1)\mathbf{x}(n-1)$

Ang & Farhang $\boldsymbol{\psi}(n) = \alpha\boldsymbol{\psi}(n-1) + e(n-1)\mathbf{x}(n-1)$

Benveniste $\boldsymbol{\psi} = [\mathbf{I} - \mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)]\boldsymbol{\psi}(n-1) + e(n-1)\mathbf{x}(n-1)$

Figure 3.3 shows the Ang & Farhang method with different values of α. If α is set too high, there runs a risk of the system becoming unstale. We can see this when $\alpha = 0.9$.

Figure 3.4 shows the three different variants. The winner appears to be the Benveniste algorithm, which overtakes everyt other algorithm to come in to steady state first. This is followed closely by the Ang & Farhang filter, and lastly Matthews & Xie. Overall, they clearly able to converge faster than the standard LMS algorithm, whilst at the same time remaining with a low steady state error.

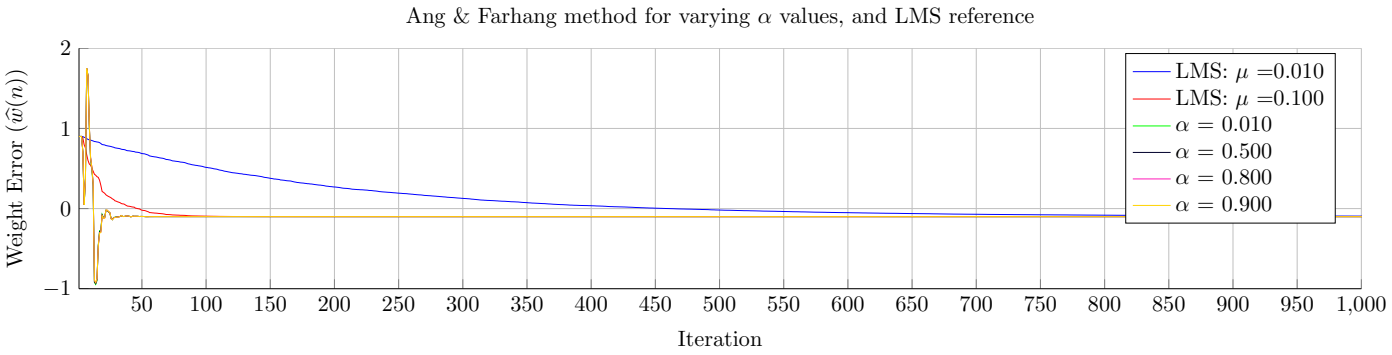


Figure 3.3: *LMS filter error (dB) for the given AR process, averaged over 100 independent trials*

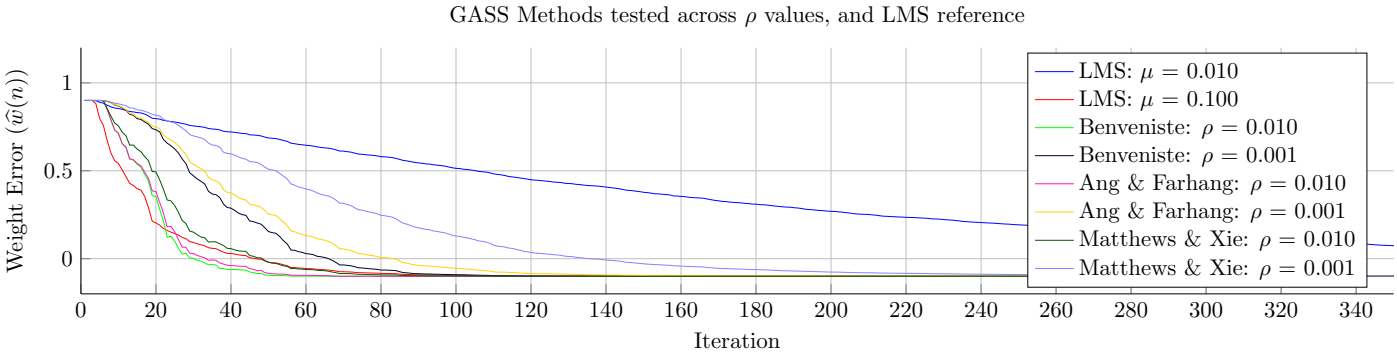


Figure 3.4: *LMS filter error (dB) for the given AR process, averaged over 100 independent trials*

3.2.b NLMS Algorithm

The NLMS algorithm is a modification to the LMS algorithm which normalises the power of the input, which for a particularly noisy (in the power spectrum) input signal means the filter can still get to a stable output, which may not happen with the standard LMS filter, or at least not without manually adjusting the step size.

Given the update equation $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n)$ and the *a posteriori* relationship given: $e_p(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1)$, we can express $\triangle\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n)$ to allow us to equate the update equation to the standard NLMS form.

First, we expand the given relationship of $e_p(n)$:

$$e_p(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1) \tag{11a}$$

$$e_p(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n) - \mu\mathbf{x}^T(n)e_p(n)\mathbf{x}(n) \qquad \text{Expanding the } \mathbf{w}(n+1) \text{ term} \tag{11b}$$

$$e_p(n)(1 + \mu\|\mathbf{x}(n)\|^2) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n) \tag{11c}$$

$$e_p(n) = \frac{e(n)}{1 + \mu\|\mathbf{x}(n)\|^2} \tag{11d}$$

We now substitute this in to the update equation:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \tag{12a}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mu e(n)\mathbf{x}(n)}{1 + \mu\|\mathbf{x}(n)\|^2} \tag{12b}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\left[\frac{1}{1 + \mu\|\mathbf{x}(n)\|^2}\right]\mathbf{x}(n) \tag{12c}$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \left[\frac{1}{\frac{1}{\mu} + \mathbf{x}^T(n)\mathbf{x}(n)}\right]e(n)\mathbf{x}(n) \tag{12d}$$

We can see that the last line, equation 12d now has the same form (and thus is equivalent to, other than the factors which are discussed below) as the NLMS update equation:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\beta}{\epsilon + \mathbf{x}^T(n)\mathbf{x}(n)}e(n)\mathbf{x}(n)$$

Thus from the comparison of forms, we can equate $\beta = 1$ and $\epsilon = \frac{1}{\mu}$.

3.2.c GNGD Algorithm

The GNGD Algorithm is a specific variety of NLMS filter. Figure 3.5 compares its performance to that of the best performing GASS algorithm we saw, the Benveniste algorithm. The GNGD significantly outperforms the Benveniste algorithm at reaching steady state.

Comparing the complexity of each algorithm, they both have vector/matrix multiplications in them. The Benvenieste has marginally more complex mutltiplications, but fundamentally, both algorithms are of order of N^2 complexity. Considering their similar computational overhead, the GNGD appears the better algorithm since it is able to reach steady state significantly faster than the Benveniste algorithm.

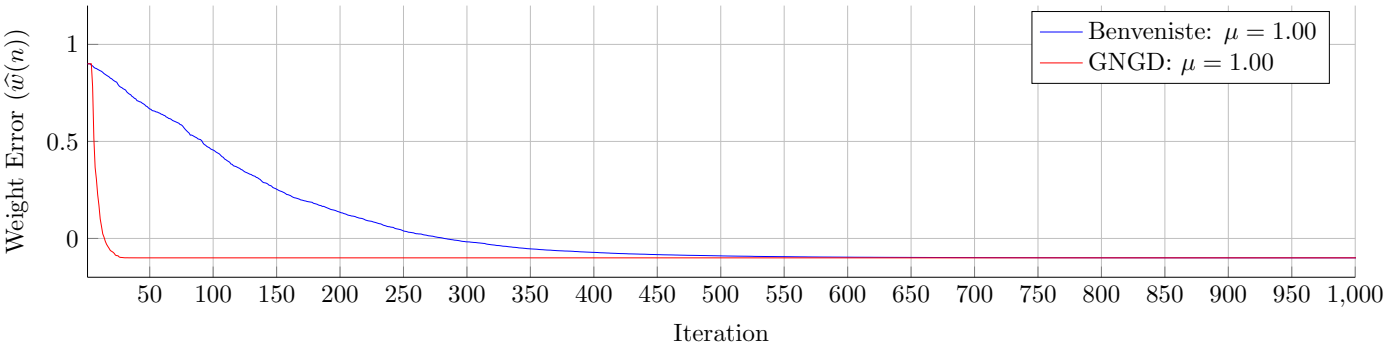


Figure 3.5: *LMS filter error (dB) for the given AR process, averaged over 100 independent trials*

3.3 Adaptive Noise Cancellation

3.3.a The Effects of Correlation on the Adaptive Line Enhancement Filter

3.3.b Parameters of the Adaptive Line Enhancement Filter

Having implemented the ALE filter, we can test two of the parameters which determine its performance: the time delay, $\triangle$, applied in the input signal which is fed in to the Linear Predictor, and the order of the LMS filter (the heart of the Linear Predictor) itself.

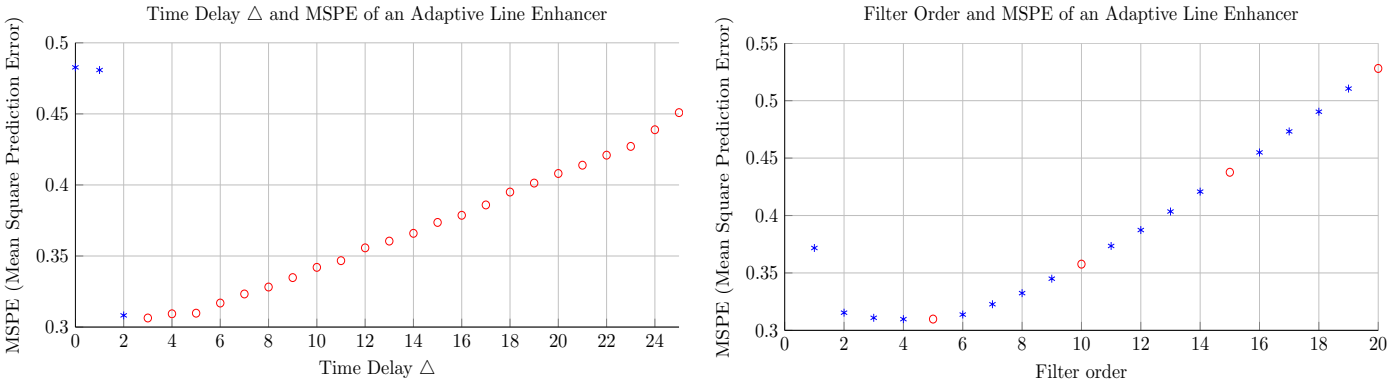
The signal $S(n)$ is composed of a sine wave and noise, such that $s(n) = x(n) + \eta(n)$ where $x(n)$ refers to the sine wave (with angular frequency $\omega_0 = 0.01\pi$) and $\eta(n) = v(n) + 0.5v(n-2)$ where $v(n) \sim \mathcal{N}(0, 1)$.

Figure **??** shows the two parameters which have been varied. In order to ensure reliable results, a 'Monte Carlo' style simulation was conducted, taking the average results across ten iterations of the data. The data remained constant between the parameter sweeps, in order to ensure a fair test. Whilst there were specific parameters within which to test the time delay $\triangle$ and filter order. But since the code was written, it was decided to explore those parameter values around and between the ones specified. The ones specified are marked on the plots as red circles, with blue stars representing other parameters tested.

Figure 3.6a shows how the MSPE changes as the time delay $\triangle$ to the filter input $\mathbf{u}(n)$ changes. We can see the lowest MSPE lies around $\triangle = 4$ and $\triangle = 6$. Unsurprisingly, $\triangle$ at low values shows very poor performance - since the noise in $s(n)$ will still be correlated with $\mathbf{u}(n)$ (as analysed in Section 3.3.a). At larger values, the MSPE also appears to get worse. By observing the output of the system, you notice that as the delay increases, the output becomes less and less in phase with the input, thus becoming less accurate not because of noise itself, but because it is out of phase with the original signal.

Figure 3.6b shows the relationship between the MSPE and filter order, M. We can see the minima of the MSPE is when $M = 5$. The Complexity of an LMS filter is $2M + 1$ [2]. Thus for every order we add to the filter, we are increasing the computational complexity of the problem. It is clear that a filter order of $M = 1$ or $M = 2$ has very poor performance, but an order of 3 or 4 may be an acceptable trade off for filter order, considering the extra computational cost. This importance of additional computational cost does depend on the device and situation though - here using desktop computers, extra parameters within the bounds of the coursework cause no discernible problem in computation time. If this were for example, on an embedded low power DSP chip, this trade off may become more apparent and need to be anaylsed in further depth. In comparison to many algorithms which are proportional to N^2 (where N reflects the size of the computational problem), a linear relationship is still fairly good. Thus we can be comfortable selecting the optimal order filter, 5, knowing there is little overhead in doing so.

Using the parameters $\triangle = 4$ and $M = 5$, we can produce a plot of the input signal without noise $x(n)$, once noise has been added, $s(n)$ and the new estimated output. As with other experiments, multiple iterations have been taken, and the mean sampled from these (as has been done in other parts of the coursework, as well). This is shown in figure 3.7. The delay on the output signal as created by $\mathbf{u}(n)$ is apparent. We also observe that the amplitude of noise is noticeably smaller on the output than the input - so we know the ALE has had some effect to improve the signal.



(a) *Differing delays plotted against their MSPEs*

(b) *Differing filter orders plotted against their MSPEs*

Figure 3.6: *MSPE of the Adaptive Line Enhancer, with varying parameters of input delay and filter order*

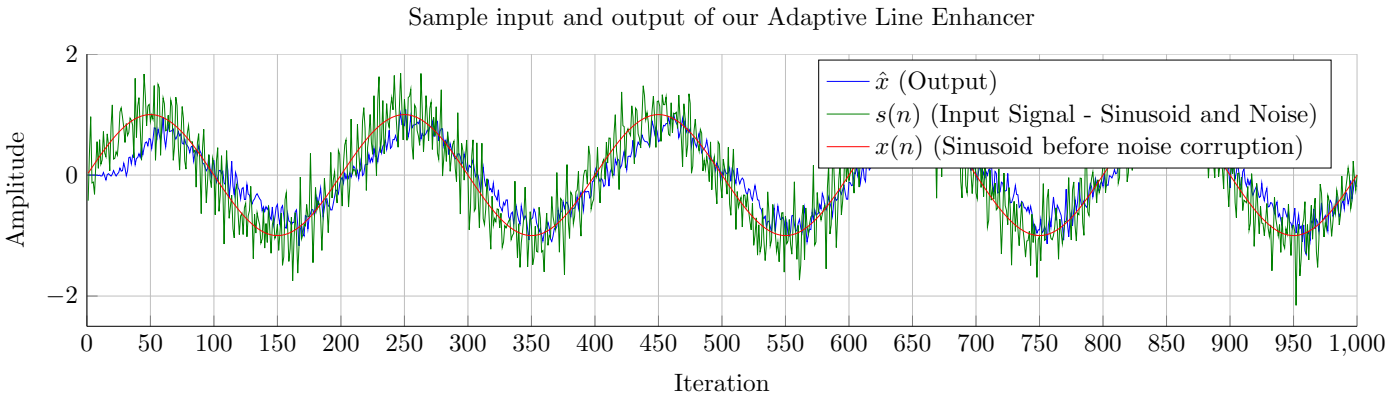


Figure 3.7: *Sample input and output of an Adaptive Line Enhancer (Averaged over 10 trials)*

3.3.c Adaptive Noise Cancellation Filter

The Adaptive Noise Cancellation is a different configuration for denoising a signal. It takes a secondary signal, $\mathbf{u}$ and uses this as the sample noise (instead of taking a delayed copy of the signal itself). In this set up, we generate another signal of White Gaussian noise (defined as $\eta(n) = v(n) + 0.5v(n-2)$ where $v(n) \sim \mathcal{N}(0,1)$.). Figure 3.8 shows the comparative Square Errors between the ALE and ANC configurations. It is apparent that the ANC outperforms the ALE. Taking the Mean Squared Error (ie across every iteration), the MSE for ALE is 1.27, whereas the ANC is 0.31.

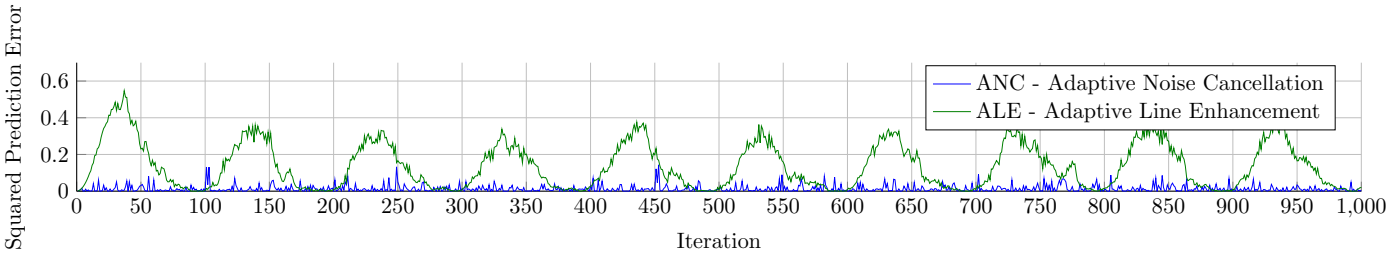


Figure 3.8: *Square Error for ANC and ALE configurations for denoising*

3.3.d Removing Mains 50Hz hum from ECG Data

The ECG data has a clear 50Hz hum, as can been seen on the spectrogram and periodogram. Using an ANC configured filter, we can generate WGN and add it to a 50Hz sine wave, to be used as the input $\mathbf{u}(n)$. We use the same spectrogram set up as earleir on, with an FFT length of 4096, 3000 sample window overlap, and the Hanning window. The spectrogram showed itself useful with that EEG data, and this is should once again prove useful here.

We will experiment with different filter lengths, as well as different step sizes. Figure 3.9 shows different step sizes used. We can see that for large values of μ, the filter is perhaps a little too agressive: for $\mu = 0.010$, there is a clear change of the 100Hz band. And for $\mu = 0.050$, the whole signal has been signficantly distorted by the filter.

Figure 3.10 shows different order filters tested out. Order selection appears to be important: too low and the noise is not fully removed. Too high and we see an injection at 100Hz (although this still happens to a minor extent), and for $M = 20$ it appears to have taken some of the signal at around the $50Hz$ band.

Based on our experimentation, we will select a filter with $\mu = 0.001$ (or even slightly lower than this), and order 10.

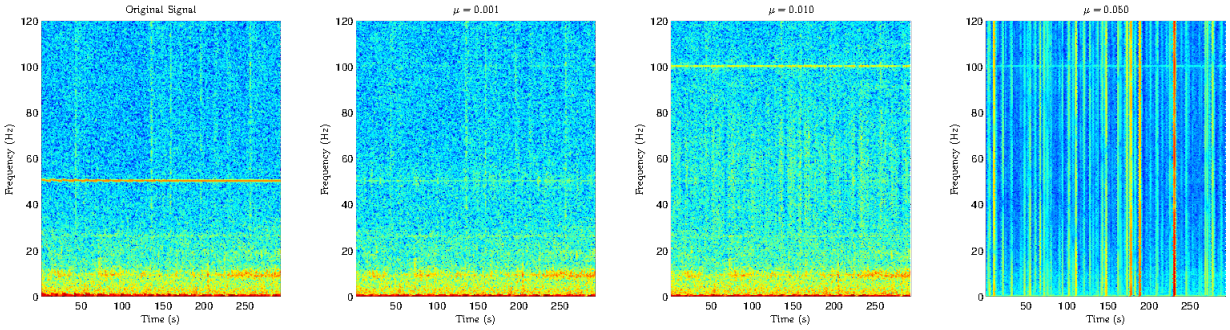


Figure 3.9: *Various levels of μ in the ANC filter, and its effect on the output*

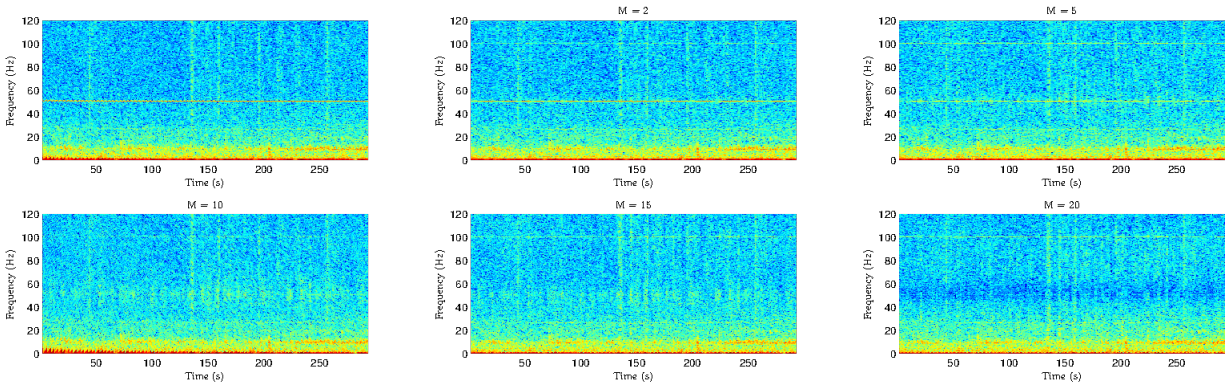


Figure 3.10: *Different model order lengths of the ANC filter, and its effect on the output*

4 Widely Linear Filtering and Adaptive Spectral Estimation

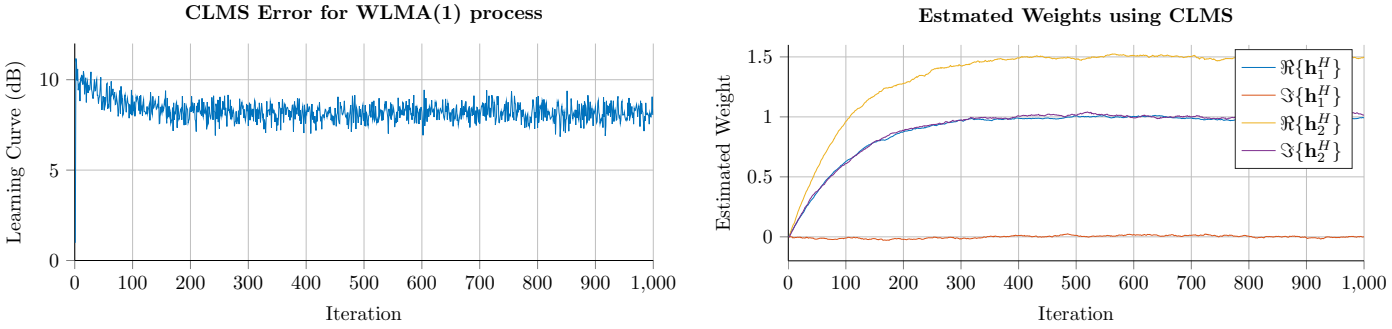
4.1 Complex LMS and Widely Linear Modelling

4.1.a The CLMS and ACLMS

We are introduced to the Complex LMS, which is a modification of the LMS filter, but designed to operate with complex signals. The $\mathbf{w}$ weight term is replaced with $\mathbf{h}$, and the estimation is now $\hat{y}(n) = \mathbf{h}^H(n)\mathbf{x}(n)$, with the weight update being performed as $\mathbf{h}(n+1) = \mathbf{h}(n) + \mu e^*(n)\mathbf{x}(n)$.

We are also introduced to the Augmented CLMS, which is designed to identify the second order (if there is any) statistical relationship of the input and output. It is effectively an extension of the CLMS, and adds the weights $\mathbf{g}$, in addition to $\mathbf{h}$ as defined for the CLMS. The esimtation equation becomes $\hat{y}(n) = \mathbf{h}^H(n)\mathbf{x}(n) + \mathbf{g}^H(n)\mathbf{x}^*(n)$. The next estimated of $\mathbf{g}$ is defined as $\mathbf{g}(n+1) = \mathbf{g}(n) + \mu e^*(n)\mathbf{x} * (n)$.

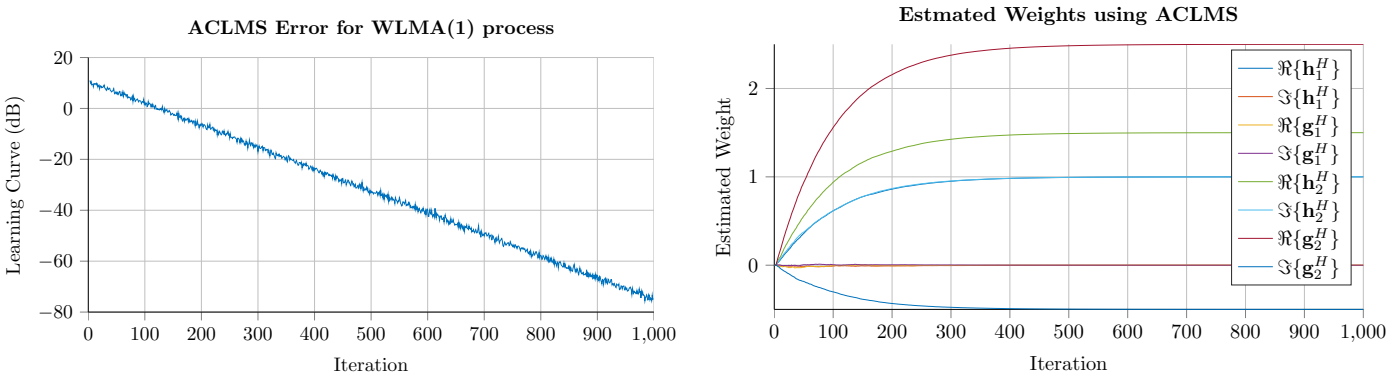
We generated a WLMA(1) (Widely Linear Moving Average) process which is defined as $y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-2)$, where $x \sim \mathcal{N}(0,1)$. The coefficients are defined as $b_1 = 1.5 + 1j$ and $b_2 = 2.5 - 0.5j$. We run this process through both the CLMS and ACLMS filters. 100 indepdent sets of noise were generated and run through both filters (identical signals, to ensure a fair comparison), with the mean of at each iteration taken. Figure 4.1a shows the learning error. We can see that even in steady state, there appears to be significant modelling error, sitting at 8dB. This is also apparent in figure 4.1b where we can see the estimated weights of $\mathbf{h}$. We are unable to see the values of coefficients related to b_2.



(a) *Learning Curve of the filter, defined as* $10\log|e(n)|^2$ **(b)** *Esimated Weights (both real and imaginary components)*

Figure 4.1: *Learning Curve and Estimated Weights from the CLMS fitler*

Figure 4.2 shows the same plots but this time from the ACLMS filter. We can immediately see that the learning curve in figure 4.2a is significantly better than those seen in figure 4.1a with the CLMS filter. Since we are modelling an WLMA process, the output can be determined directly from the input (unlike with a 'next step' estimator), so we expect the error to be very small. This is also evident in the weights in figure 4.2b. We also observe that we can see 4 weights (the real and imaginary components of both $\mathbf{g}$ and $\mathbf{h}$), as opposed to the two seen in the CLMS. We can observe that the values associated with b_2 are represented by $\mathbf{g}$.



(a) *Learning Curve of the filter, defined as* $10\log|e(n)|^2$ **(b)** *Estimated Weights (both real and imaginary components)*

Figure 4.2: *Learning Curve and Estimated Weights from the ACLMS fitler*

We can conclude that the CLMS was unable to represent the WLMA properly - the coefficients of b_2 which were from the conjugate of $\mathbf{x}$ were nowhere to be seen in the CLMS, but very clear in the ACLMS. The fact that the learning curve of the CLMS showed a large steady state error is evidence that the model was a poor fit for the CLMS.

4.1.b Bivariate Wind Data

We are given wind data for a slow, medium and high wind regime. The data contains wind East-West wind speeds, as well as North-South wind speeds for a given point in time. We represent these in complex form, as $v[n] = v_{east}[n] + jv_{north}[n]$. Figure 4.3 shows the circularity plots for all three regimes of wind. All of the circularity plots appear to be fairly circular. The low wind seems to have a narrower Real set of data than it does in the Imaginary plane. The high wind appears to be shofted off centre, but also has significantly higher amplitude than the low wind (as we would expect!). It appears to have some correlation, observed on the positive real and negative imaginary axes.

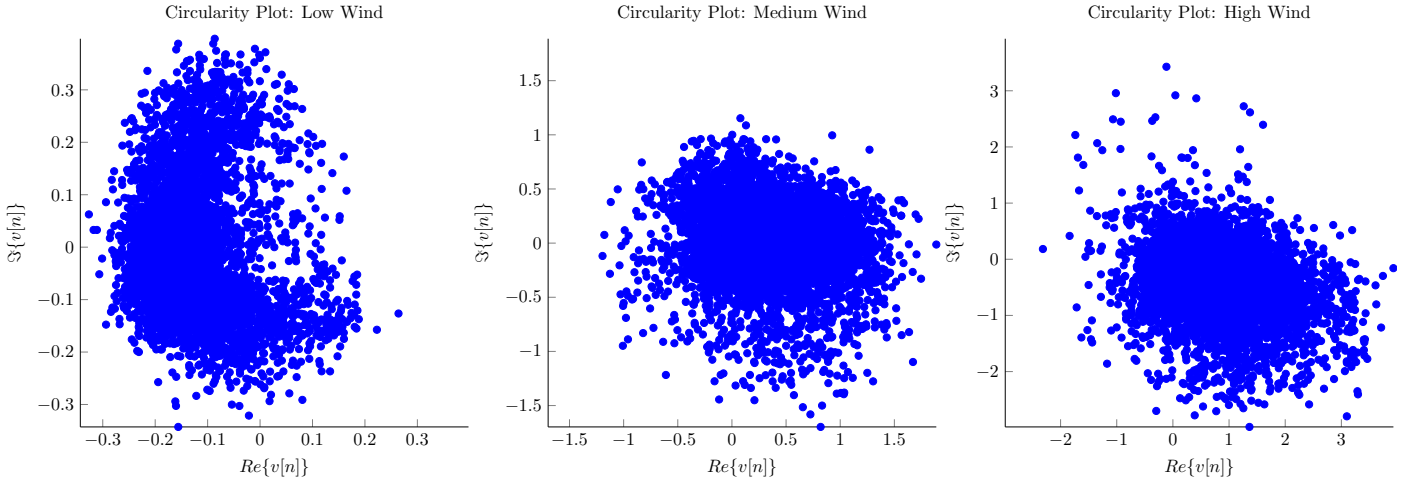


Figure 4.3: *Circularity Plots for Wind Data*

With the 5000 data points for each wind regime, we can test out filters to predict the next state of the wind. Figure 4.4 shows the different regimes tested against both the CLMS and ACLMS filters, with varying model orders. As defined in Section 3.3.a, we use the Mean Squared Prediction Error to find the optimal filter and order. Although there will be some error during setup as the filter coefficients initialise, this is common to all filters and thus will be considered part of the error. The Low Wind demonstrates good performance, with an MSPE of -19dB at a minimum, with it tending to -22dB as the order increases. The ACLMS appears to outperform the CLMS, although only marginally. Medium Wind has a minimum (ie optimum filter length) at 4, using the ACLMS. It obtains an MSPE of around -12.3dB. Interestingly, the CLMS outperforms the ACLMS at values higher than this. The high wind is interesting, as for high model orders the MSPE is very large (hundreds of dB). With an ACLMS model order of 1, an MSPE of -6.25dB is acheived, which represents reasonable performance, especially when considering what happens with a larger order filter!

In conclusion, we are able to model all three systems effectively (all with an MSPE of -6dB or better) in a next value predictor set up, using widely linear filtering. We find that in all three cases the ACLMS does outperform the CLMS, at least for the optimum values (the minimum of these plots) which we are interested in.

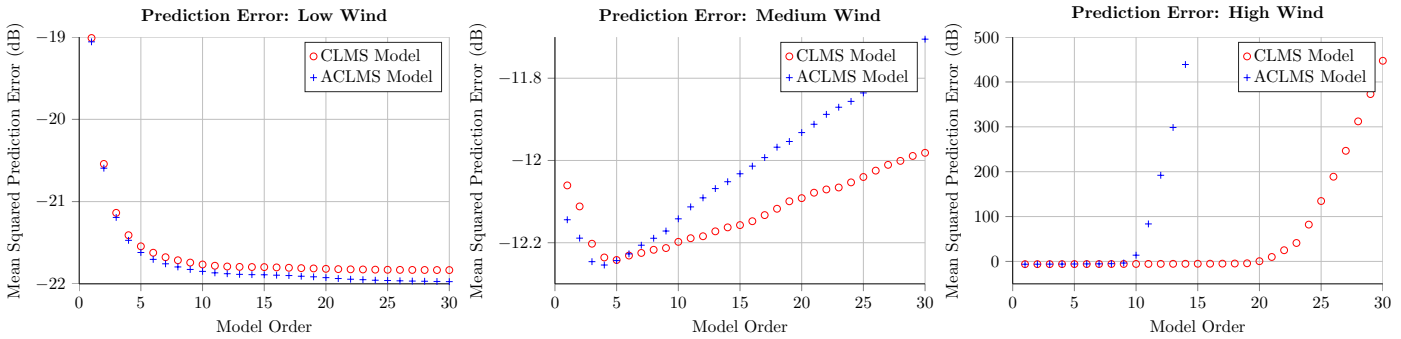


Figure 4.4: *Prediction Error Estimates for varying Model Orders*

4.1.c Three Phase Power: Balanced & Unbalanced Systems

Using the Clarke Voltage, we can compute $v(n)$ for both a balanced and unbalanced system. Figure 4.5 shows the circularity diagram of the complex voltage (both a balanced and unbalanced system). The balanced system has a circular shape to it, whereas the unbalanced system's circularity diagram is more akin to a rugby ball than a circle. The unbalanced nature of the system was demonstated when either a phase shift is introduced in to one or two of the lines, or the voltage across each phase is different. In this case, a phase shift of $\frac{\pi}{8}$ has been applied to one phase, with another having phase shift $-\frac{\pi}{7}$.

Assuming a fault on the system will either cause a disturbance in voltage levels, or shift the phase, its circularity plot would start no longer looking like a circle, but instead take a distorted shape like that observed in Figure **??** for the

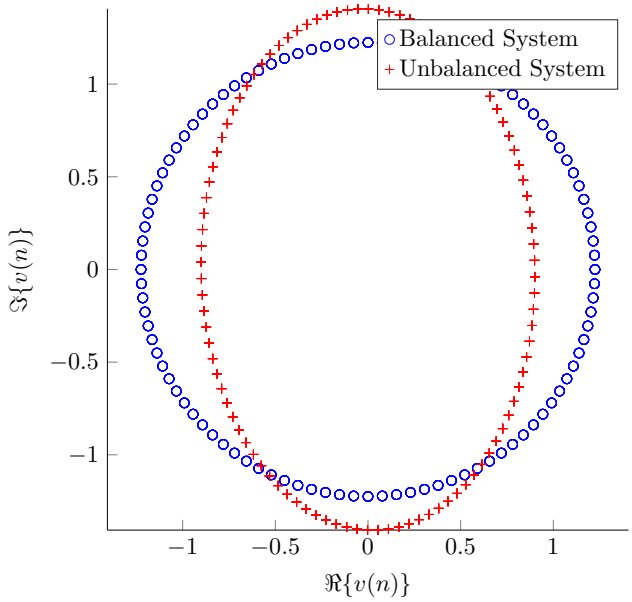


Figure 4.5: *Balanced and Unbalanced Complex Voltages*

unbalanced system.

4.1.d Three Phase Power: Widely Linear AR models

We are given the strictly linear and widely linear AR(1) models used on three phase power systems, and from this we are able to determine the frequency of the Clarke Voltage.

For the strictly linear case, we know the model is defined as $v(n+1) = h^*(n)v(n)$. Using the definition of the Clarke voltage for a balanced system [[5]], we know that

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n + \phi)} \tag{13}$$

We can equate these two equations together in order to determine the operating frequency from the weights of the CLMS model:

$$\begin{aligned} v(n+1) &= h^*(n)v(n) \\ e^{j(2\pi \frac{f_o}{f_s}(n+1)+\phi)} &= h^*(n)e^{j(2\pi \frac{f_o}{f_s} n+\phi)} \\ e^{j(2\pi \frac{f_o}{f_s})} &= h^*(n) \\ 2\pi \frac{f_o}{f_s} &= \arctan\left(\frac{-\Im\{h(n)\}}{\Re\{h(n)\}}\right) \end{aligned} \tag{14}$$

Note: the equations in the coursework handout appear not to have taken in to account the negative conjugate, which is why if applied in `MATLAB` exactly as here, they will result in producing a negative frequency.

The unbalanced system is a slightly more involved derivation. Firstly we use the definitions of the Clarke Voltage and Widely Linear models (respectively) for an unbalaced system:

$$v(n) = A(n)e^{j(2\pi \frac{f_o}{f_s} n+\phi)} + B(n)e^{-j(2\pi \frac{f_o}{f_s} n+\phi)} \tag{15}$$

$$v(n+1) = h^*(n)v(n) + g^*(n)v^*(n) \tag{16}$$

We can then substitute these together

$$\begin{aligned} A(n+1)e^{j(2\pi \frac{f_o}{f_s}(n+1)+\phi)} + B(n+1)e^{-j(2\pi \frac{f_o}{f_s}(n+1)+\phi)} = h^*(n)A(n)e^{j(2\pi \frac{f_o}{f_s} n+\phi)} + B(n)e^{-j(2\pi \frac{f_o}{f_s} n+\phi)} + \\ g^*(n)A^*(n)e^{-j(2\pi \frac{f_o}{f_s} n+\phi)} + B^*(n)e^{j(2\pi \frac{f_o}{f_s} n+\phi)} \end{aligned} \tag{17}$$

4.1.e Estimating the Frquency

Based on the Strictly and Widely linear filters, we can estimate the frequency. Based on the data produced in section 4.1.c, we can reverse engineer the system frequency.

For the case of the balanced system, the strictly linear equation returns $-50Hz$ as the system frequency. As mentioned in 4.1.d, this is likely due to a conjugate error during derivation. The system also identifies the unbalanced system as

$50.051Hz$. This was measured using the widely linear input. If we use the strictly linear equations, we are returned the frequency $-50Hz$ (or $50Hz$). This will be because the stringly linear system does not have the weights and parameters (namely $\mathbf{g}(n)$) to model the unbalanced system correctly.

4.2 Adaptive AR Model Based Time-Frequency Estimation

4.2.a Stationary Estimation

Using the Yule Walker eqations and the `MATLAB` function `aryule`, we can try and estimate an AR(1) model. This clearly will be difficult, since the signal is non stationary. Thus the model for the first part of the signal will likely not be valid for the second part of the signal etc. Figure 4.6 does show the AR(1) attempt to fit to the signal, but it will likely fail across the entire time of the signal (as it goes through different stationary outputs).

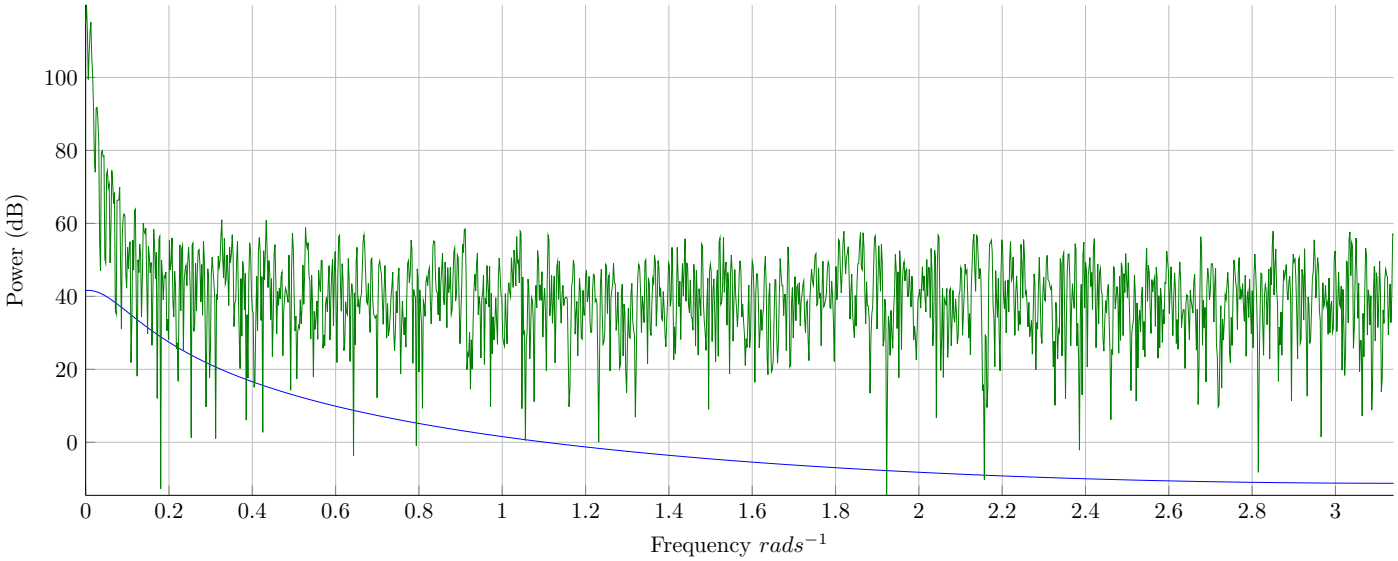


Figure 4.6: *Measured PSD (Green) and Modelled PSD (Blue)*

4.2.b CLMS Moving Average AR Estimation

Instead of using the Yule Walker equations, we have instead used the CLMS algorithm to try and model coefficients. A 3D 'surf' plot has been produced, and is in Figure 4.7 - similar to a spectrogram, modelling the coffeicients at each iteration. Note how the model changes over time, indicating that the model is non-stationary. It is for this reason that the Yule Walker modelling would not be sufficient to capture this non-stationary process.

4.3 A Real Time Spectrum Analyser Using Least Mean Square

4.3.a DFT-CLMS

The DFT-CLMS was implemented using the CLMS filter, and the data from the estimated AR coefficients passed on. Figure 4.8 shows the DFT-CLMS and esimated frequency output. It clearly does not appear quite like the spectrogram previously seen. Fundamentally, the DFT runs with a set of data each sample contributes in some fashion to each frequency bin. The CLMS algorithm runs in sequential order and thus is unable to explore each time sample to update each weight in the same fashion the DFT is.

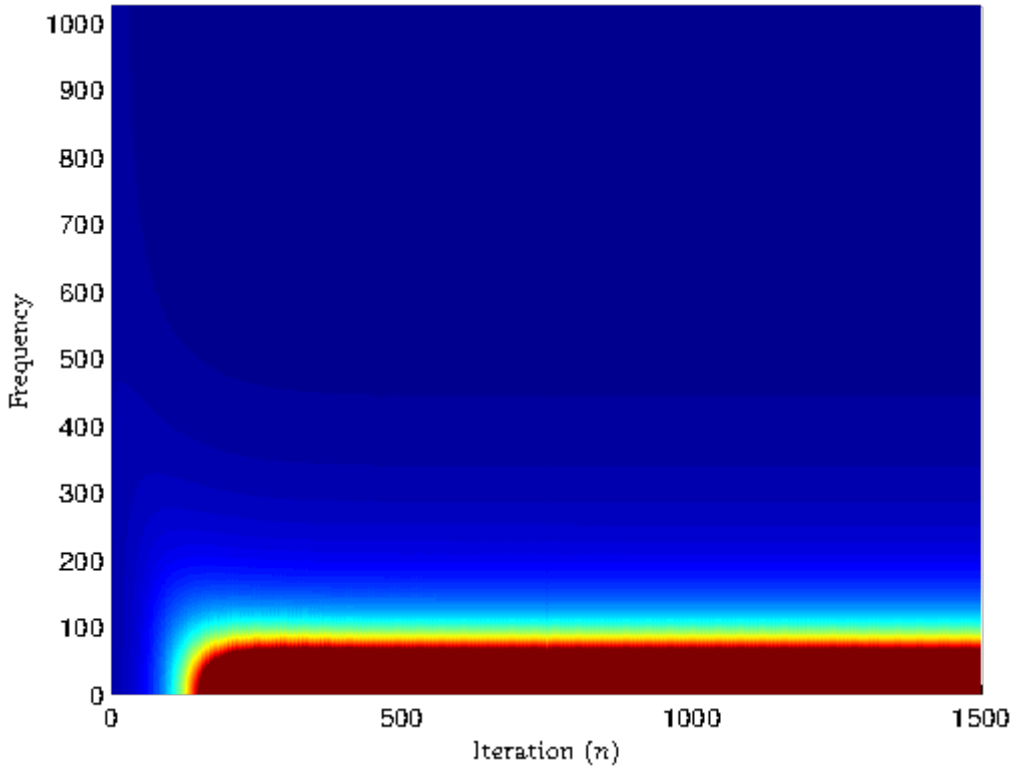


Figure 4.7: *3D graph of AR coefficients*

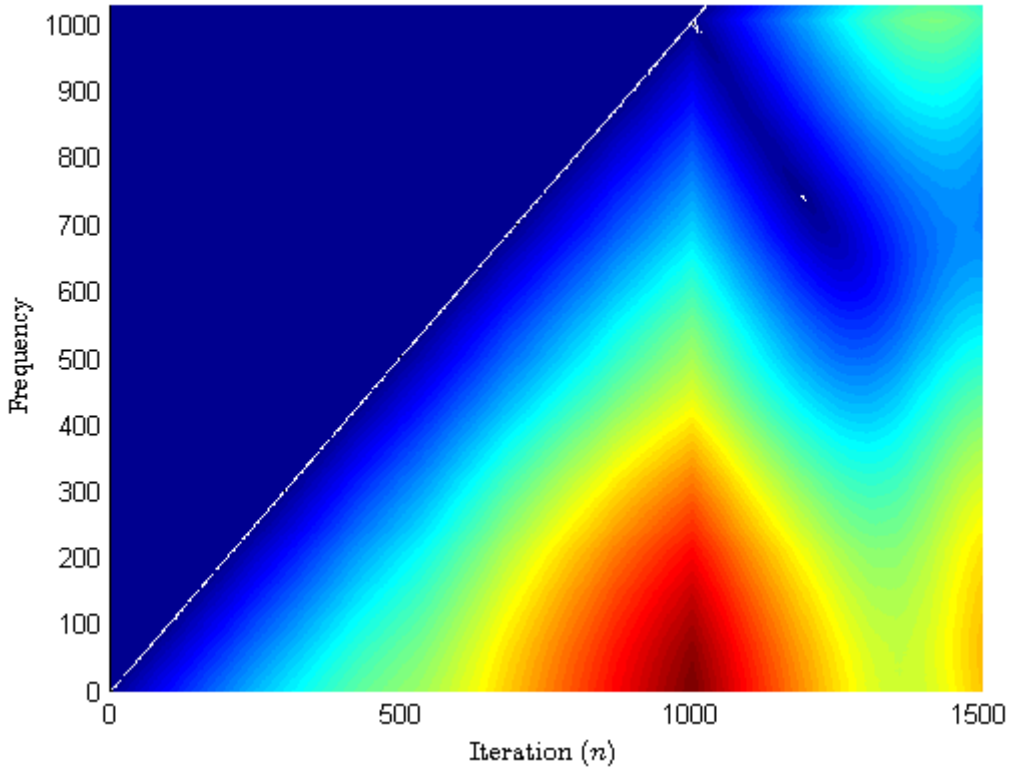


Figure 4.8: *DFT-CLMS of spectrum from the AR estimator*

References

[1] a Berger and 2001. "The Matrix Cookbook". In: *Citeseer* (2007), pp. 1–66. ISSN: 09621083. DOI: 10.1111/j.1365-294X.2006.03161.x. URL: http://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:Statistical+machine+learning+for+information+retrieval\#5.

[2] Jyoti Dhiman, Shadab Ahmad, and Kuldeep Gulia. "Comparison between Adaptive filter Algorithms (LMS , NLMS and RLS)". In: 2.5 (2013), pp. 1100–1103.

[3] Max Kamenetsky and B. Widrow. "A variable leaky LMS adaptive algorithm". In: *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.* 1 (2004). ISSN: 10586393. DOI: 10.1109/ACSSC.2004.1399103.

[4] Danilo P Mandic. *Spectral Estimation and Adaptive Signal Processing*. 2014.

[5] Danilo P Mandic. "Spectrum Estimation and Adaptive Signal Processing Coursework". In: (2015), pp. 1–20.