IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

COURSEWORK REPORT

Adaptive Signal Processing and Machine Intelligence

Author
Rohan TANGRI
01198239

April 10, 2020

Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

One definition of the PSD directly measures the average signal power across frequencies. Starting from this definition:

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

Taking the given assumption that the covariance sequence $r(k)$ decays rapidly, that is:

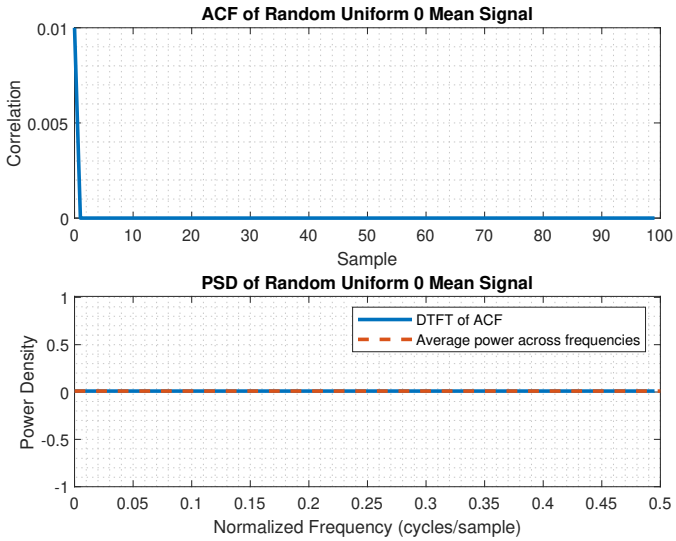
$$\lim_{n\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k||r(k)| = 0 \tag{2}$$

We can obtain the second definition of the PSD, the DTFT of the ACF:

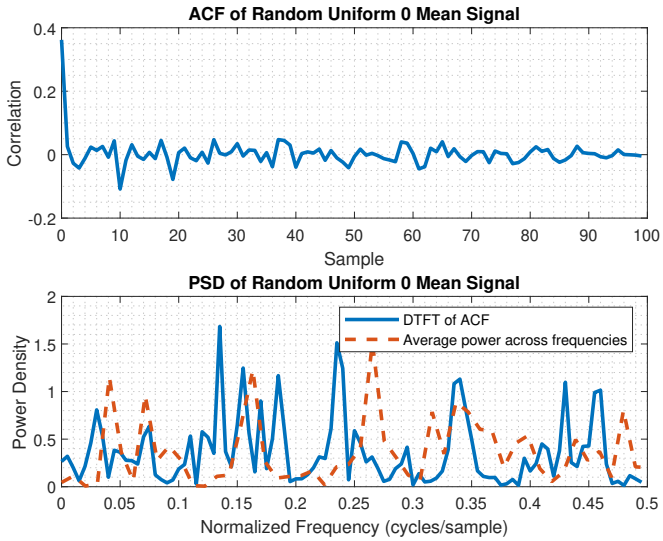
$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k) e^{-j\omega k} \tag{3}$$

The autocorrelation function itself describes how a signal is correlated with itself for different time lags. In fact, the second definition of the PSD holds so long as the signal is a Wide Sense Stationary process; that is, it has a constant mean and an autocorrelation function that only depends on the time difference; so $r(t_1, t_2) = r(t_1 - t_2)$. This relationship between the autocorrelation function and PSD is called the Wiener-Khinchin Theorem.
The autocorrelation is an even function. This makes sense since the correlation of a signal at time t_2 with respect to the signal at time t_1 should be the same as the correlation between the signal at time t_1 with respect to the signal at time t_2. For the remainder of this report, all autocorrelations (and by extension, PSDs) will only be plotted for positive time (and frequency).
Furthermore, the DFT of an even function should output a purely real signal. Therefore, the FFT of the autocorrelation function should give a real valued power spectrum. However, due to the way in which computers handle floating point numbers, the imaginary parts do not completely cancel out and so the PSD output is still complex. Therefore, taking the real part of the FFT is done to forcefully ignore the imaginary part.

The relationship between both definitions of the PSD is shown below:



(a) Plot of the ACF and PSD of an impulse signal



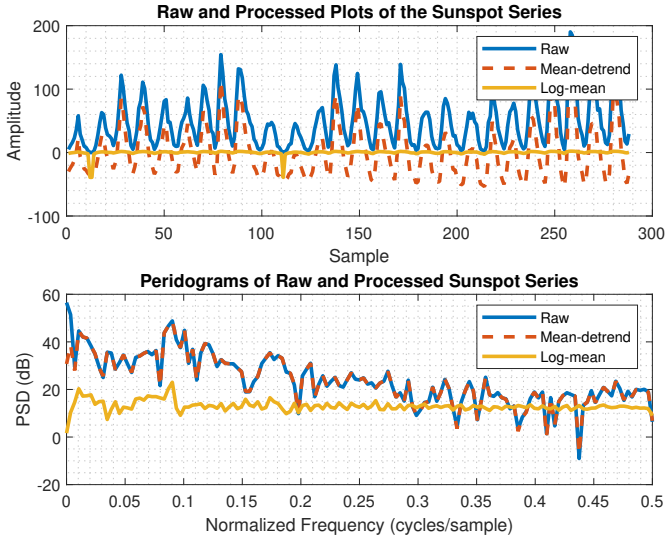
(b) Plot of the ACF and PSD of a randomly generated uniform signal

Figure 1

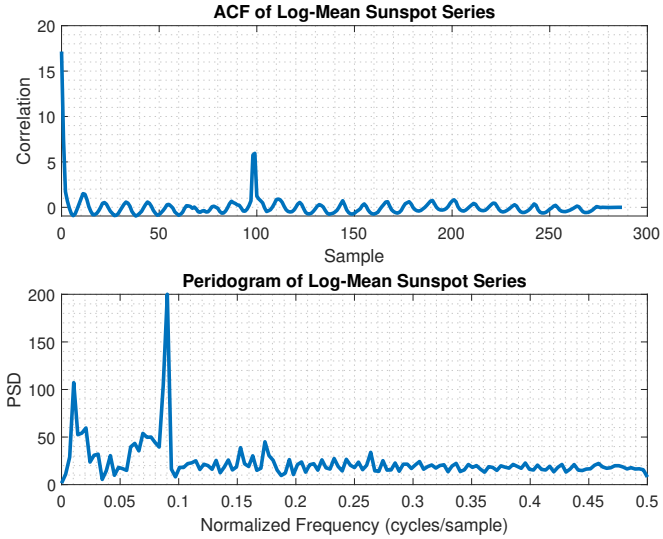
The impulse signal in Figure 1a has a rapidly decaying autocorrelation function so both definitions of the PSD give the same results. However, the randomly uniform generated zero mean signal in Figure 1b does not have a rapidly decaying autocorrelation function so the relationship between both PSD functions breaks down.

1.2 Periodogram-based Methods Applied to Real-World Data

Below is illustrated the sunspot time series and its spectral estimate:



(a) Time Series and Periodogram Plots of the Sunspot Series



(b) Autocorrelation and Periodogram of log-mean Sunspot Series

Figure 2

From Figure 2a, it can be observed that the mean-detrend periodogram has a much lower power density at the zero frequency. Therefore, removing the mean and trend in the data appears to eliminate some low frequency components. This makes sense since subtracting the mean centers the time series around 0, removing the DC component at $f = 0$, whilst the detrend function subtracts the best-fit line (in the least-squares sense) from the time series.
From Figure 2b, the periodogram of the log-mean sunspot series indicates two main peaks at frequencies 0.01 cycles/sample and 0.09 cycles/sample. The peak at 0.01 cycles/samples indicates a periodicity of $\frac{1}{0.01} = 100$ samples. This

matches the autocorrelation plot which shows a peak at a lag of 100 samples. However, the periodogram peak at 0.09 cycles/sample indicates a periodicity of $\frac{1}{0.09} = 11$ samples which is not represented by any significant peak on the autocorrelation diagram.

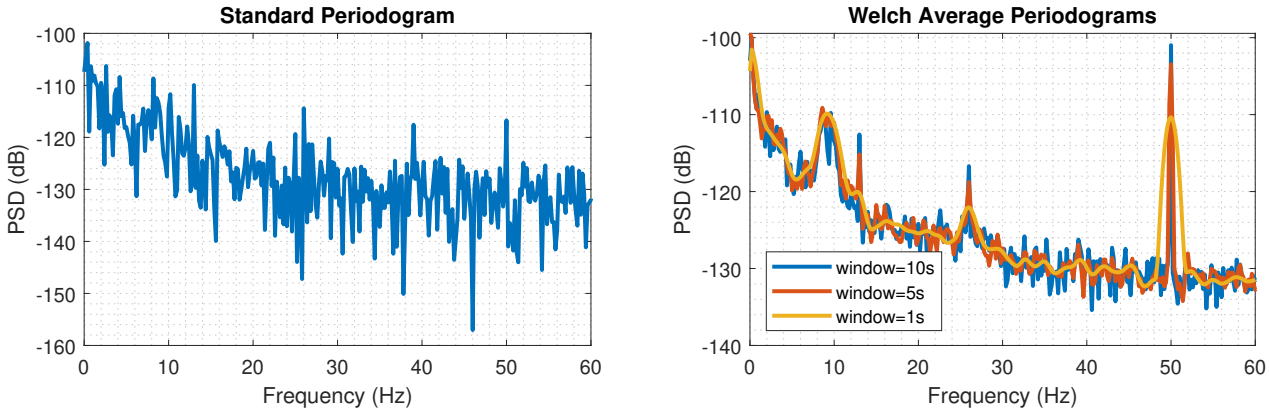


Figure 3: Periodograms of EEG Data

Figure 3 compares the performance of a standard periodogram with the Welch average periodogram of different window lengths. The Welch method aims to reduce the variance of the PSD. To do this, the data is first split into possibly overlapping segments before taking the DFT of each segment. Each individual output periodogram is then averaged to obtain the final reduced variance periodogram.
There is a trade-off between the variance and frequency resolution as the window length is changed. A shorter window length means more segments to average together, leading to a reduced PSD variance. However, the cost of this is a reduction in frequency resolution, so it is more difficult to distinguish closely spaced narrow-band components.
This is verified in Figure 3. In all four estimates, the SSVEP response can be identified by a peak at $f_{SSVEP} = 13Hz$ with a clearly defined harmonic, $f^{h1}_{SSVEP} = 26Hz$ and another identifiable but less easily distinguishable harmonic, $f^{h2}_{SSVEP} = 39Hz$, the alpha-rhythm can be identified between 8 to 10Hz and the power-line interference at $f_{PL} = 50Hz$ can also be clearly distinguished. It should be noted that all of the Welch periodograms appear smoother than the standard periodogram, and as the window length decreases, this smoothing effect only increases. The payoff here is that the peaks become much wider and overall fewer peaks can be distinguished. In this case, a window length of 5s appears to give the best results where $f_{SSVEP} = 13Hz$ is most clearly distinguishable from the surrounding spectrum.

1.3 Correlation Estimation

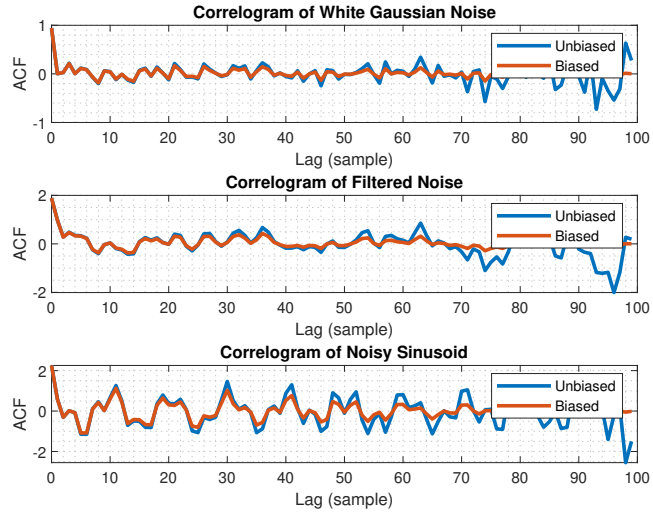
Thus far, the biased sample autocorrelation has been used. The equations for the raw autocorrelation, biased autocorrelation and unbiased autocorrelation estimators are given below:

$$R_{xy}(m) = \sum_{t=k+1}^{T} (x_t - \bar{x})(x_{t-k} - \bar{x}) \tag{4}$$

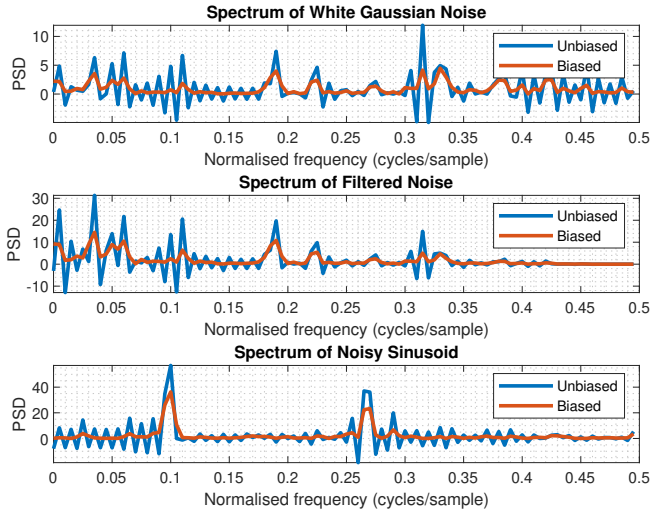
$$R_{xy,biased}(m) = \frac{1}{T} R_{xy}(m) \tag{5}$$

$$R_{xy,unbiased}(m) = \frac{1}{T - |k|} R_{xy}(m) \tag{6}$$

The biased estimator essentially bounds the variance by applying a Bartlett window to the unbiased autocorrelation, that is $R_{xy,biased} = (T - |k|)R_{xy,unbiased}$. This has the effect of smoothing the power spectrum which is necessary for statistical stability and down-weights the less reliable large-lag estimates.



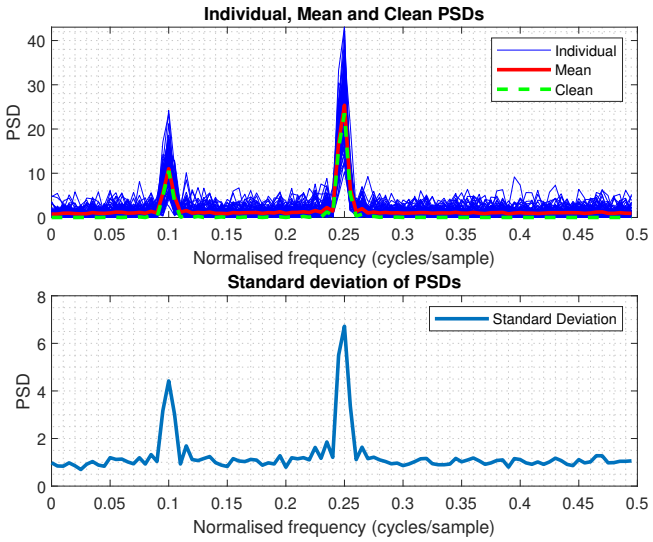
(a) Unbiased vs Biased Correlograms



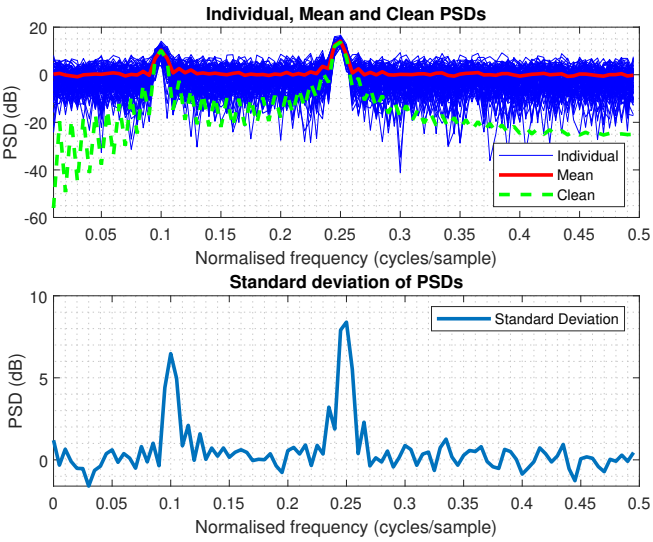
(b) Unbiased vs Biased Spectra

Figure 4

Figure 4a illustrates how the biased estimator smoothes the autocorrelation function of a signal, especially at large lags, a desirable effect. Moreover, the correlations at low lags are very similar for both estimates, in this case, below a lag of $k = 30$ samples. Furthermore, Figure 4b shows some negative PSD values for the spectra computed using the unbiased estimator and an overall smoother curve for the biased spectra. The biased estimator does not suffer from negative PSD values since the Fourier transform of the Bartlett window is strictly non-negative. The combination of these effects promotes the biased estimator as the better alternative.



(a) PSD Statistics



(b) PSD Statistics with dB scaling

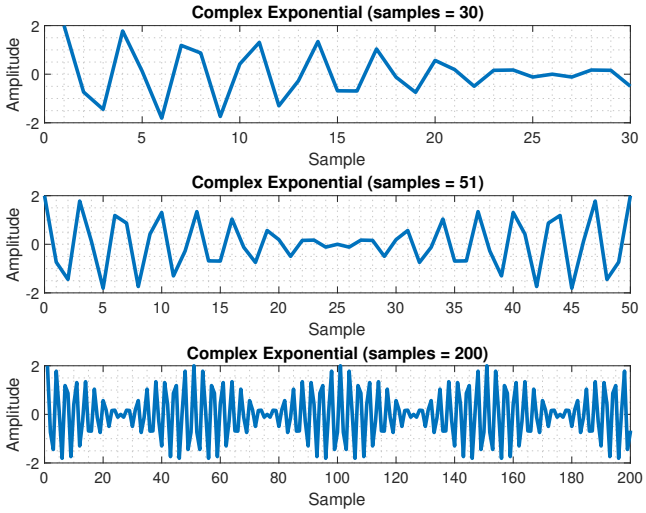
Figure 5

The upper graph in Figure 5a shows the spectrum of the signal:

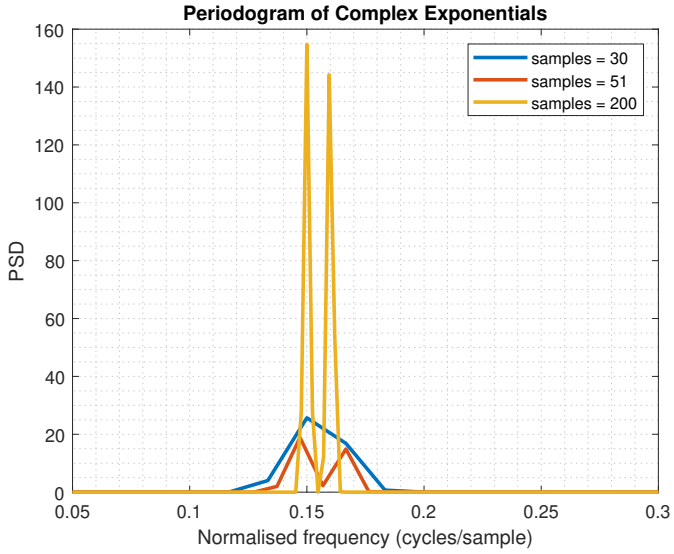
$$x(n) = 1.5sin(2\pi * 0.3n) + sin(2\pi * 0.6n) + 2sin(2\pi * 0.9n) + w(n) \tag{7}$$

Where $w \sim \mathcal{N}(0,1)$. From the spectrum, the two sine waves can clearly be distinguished by the two peaks and furthermore the PSD of the mean of all the corrupted signals very closely matches the PSD of the clean signal with no noise. The lower graph indicates that the standard deviation of the corrupted signal is also proportional to the PSD, so there is most variance at the frequencies of the sin waves.
The dB scaling in Figure 5b exaggerates differences at very low PSD values whilst maintaining overall trends. Therefore, the clean and mean plots of PSD now look very different, with the clean plot having visibly much more variation at

frequencies not defined by the sine waves. This makes sense, since the clean wave should have a much lower (theoretically zero) PSD value at these values than the noisy wave. Therefore, the dB scaling is useful when examining low PSD variations.



(a) Real part of complex exponential signals



(b) Peridograms of different length complex exponentials

Figure 6

The periodograms in Figure 6b are for the complex signal:

$$z = e^{j(2\pi 0.3t)} + e^{j(2\pi 0.32t)} \tag{8}$$

This represents component signals with frequencies of $f_z = [0.3Hz, 0.32Hz]$. The frequency resolution of a periodogram is given by $f_{res} = f_s/N$. For the case in Figure 6, the sampling frequency is normalized such that the frequency resolution becomes $f_{res} = 1/N$. Therefore, this equation indicates that at least 50 samples are required to differentiate the two frequencies present in the signal. Figure 6b supports this claim since the periodograms of the signals of length 51 and 200 samples have clearly defined peaks representing the two frequencies whereas the signal of length 30 samples does not. Furthermore, the 200 sample periodogram has better defined peaks than the 51 sample periodogram. Another interesting fact to note is at the peaks, the PSD values decrease going from 30 to 51 samples but then increase going to 200 samples. This all makes sense from an intuitive point of view. At 30 samples, one whole period has not been completed yet. Therefore, the PSD spectrum is not representative of the whole signal. At 51 samples, a period of the wave is constructed. Therefore, the periodogram can now differentiate the two dominant frequencies present in the signal. However, the PSD values at the peaks decreases since the added energy from the increased length of the signal is not enough to counter the increased number of bins dividing the PSD values into smaller sections. At 2000 samples, there are a few periods of the signal now available. Therefore, the periodogram can now better isolate the frequencies in the signal. The values of the PSD at the peaks are much larger now because there are so many more periods of the signal, so there is a multiplier on the energy present at each frequency that overcomes the increased number of frequency bins.
Another way to increase frequency resolution without needing a longer specific signal is spectral interpolation. This is a method whereby zero padding a time domain signal introduces new frequency bins between the existing bins.

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

Figure 7

The first line of the Matlab code in Figure 7 returns the biased estimate of the symmetric autocorrelation matrix to R for input signal x of length n. The output X is a rectangular Toeplitz matrix such that $X'X = R$. The second input to the function, m, determines the size of the autocorrelation matrix, of course this cannot be larger than the length of the input n. The final input determines the method to compute X; 'mod' defines X as the 2(n-m)-by-(m+1) modified rectangular Toeplitz matrix that generates an autocorrelation estimate for the input x, derived using forward and backward prediction error estimates, based on an mth-order prediction error model.

The second line implements the multiple signal classification (MUSIC) algorithm. This works by performing eigenspace analysis of the input autocorrelation matrix, R. The output pseudospectrum is contained in the output S while F returns the frequencies of the spectrum. The first input is the autocorrelation matrix. The second input specifies the signal subspace dimension. The third input specifies the normalized frequencies for which the pseudospectrum is computed at; if this has fewer than 2 elements it is interpreted as the number of FFTs to implement. The fourth input specifies the sampling frequency. The last input defines whether the first input is an autocorrelation matrix or a raw signal.The MUSIC algorithm is defined mathematically below:

$$P_{MUSIC}(f) = \frac{1}{e^H(f)(\sum_{k=p+1}^{N} v_k v_k^H)e(f)} \tag{9}$$

Where v_k is the kth eigenvector of the correlation matrix, N is the dimension of the eigenvectors, p is the dimension of the signal subspace and $e(f)$ consists of complex exponentials. This algorithm is especially adept at processing signals that are the sum of sinusoids with additive white Gaussian noise. This is because in the eigenvalue decomposition of the autocorrelation matrix of such a signal, the largest p eigenvectors span the subspace of the signal plus noise and the remaining span the noise subspace only. Therefore, the eigenvectors in the sum span the subspace of the noise only. This amounts to summing the squared magnitudes of the FFT for each noise spanning eigenvector. This technique has better resolution and is more robust in the presence of noise than the FFT; however, it is a lot more computationally expensive with a time complexity of $O(N^3)$ vs $O(Nlog_2N)$ and it requires knowledge of the number of components within a signal prior to its application.
The third line plots the pseudospectrum and limits the frequency representation between 0.25Hz and 0.4Hz.

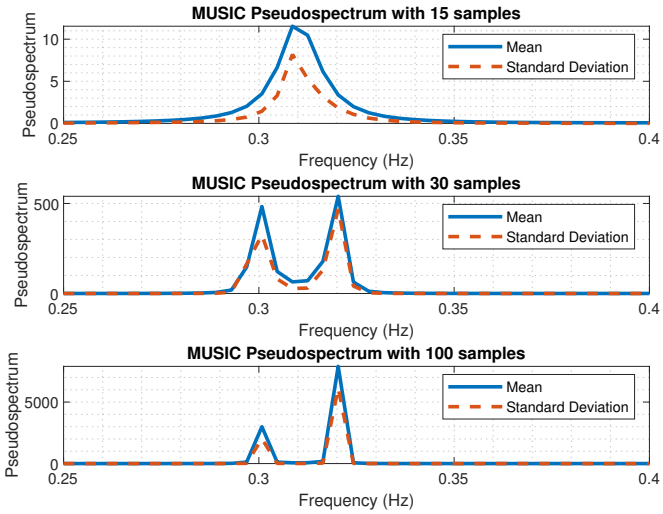


Figure 8: Mean and standard deviation of MUSIC spectrum estimation of noisy complex exponentials

Figure 8 shows the standard deviation and mean of the MUSIC spectrum for the signal (8) with added Gaussian noise. In each sample case, the standard deviation is highly correlated with the mean. At 15 samples, the MUSIC algorithm is unable to distinguish between the two signal frequencies; however, at 30 and 100 samples, the distinction is clear. This is an improvement over the standard FFT method which required a signal of at least 50 samples to obtain adequate frequency resolution.

1.4 Spectrum of Autoregressive Processes

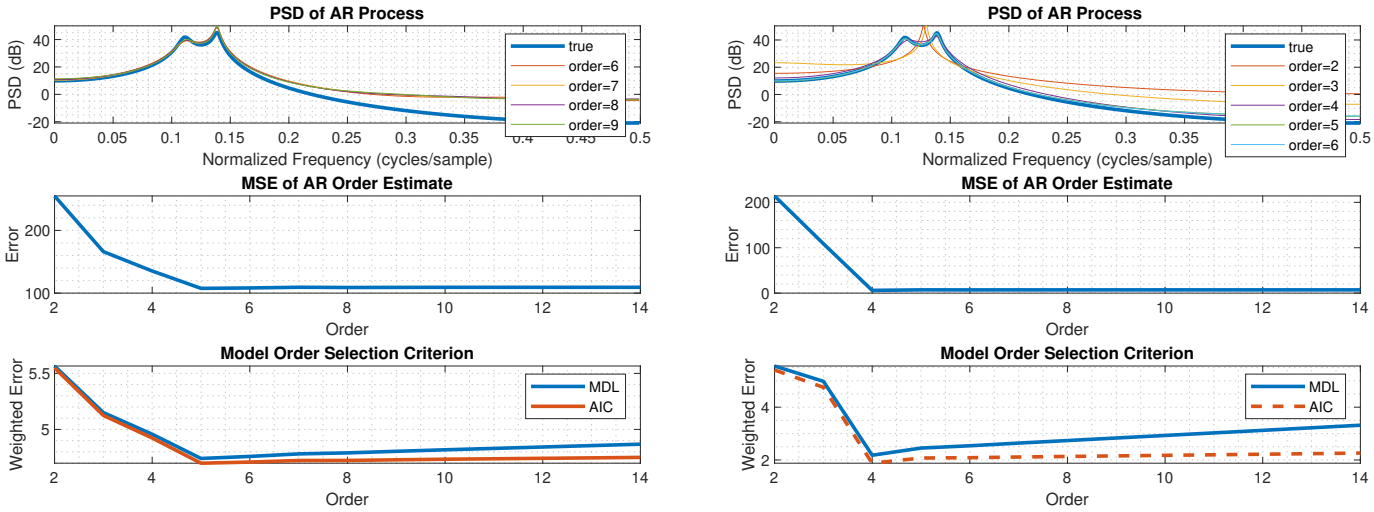
For the Yule-Walker equation to hold when finding the AR parameters, the autocorrelation matrix must be invertible. The relationship desired is shown below:

$$\boldsymbol{a} = \boldsymbol{R}^{-1}\boldsymbol{r} \tag{10}$$

$\boldsymbol{a}$ = AR coefficients vector
$\boldsymbol{R}$ = autocorrelation matrix
$\boldsymbol{r}$ = Kronecker delta function multiplied by the noise variance

The autocorrelation matrix by definition must be symmetric. A sufficient condition for a symmetric matrix to be invertible is positive definitiveness. When the autocorrelation matrix is defined as biased, this condition is met; however, if unbiased

the condition may no longer be met. Therefore, the unbiased estimate for the ACF is not suitable when finding the AR coefficients.



(a) Number of samples = 500

(b) Number of samples = 9500

Figure 9: AR process estimation

Figure 9 shows the PSD of different orders of AR estimates for a real AR(4) process. The true signal is given by the equation below:

$$x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n) \tag{11}$$

Where $w(n)$ represents a standard normal distribution. From the MSE plots, it can be observed that as the order of the AR model increases, the error decreases until it plateaus at some order. At low order estimates, there are not enough degrees of freedom to fit the signal appropriately. At higher order AR estimates, despite having a low error, the higher order coefficients are close to zero and do not contribute significantly to the estimate. This wasted complexity is often referred to as over-fitting. Therefore, There is a trade-off between complexity and model accuracy which can be measured by the Minimum Description Length criterion (MDL) or the Akaike Information Criterion (AIC). The equations for both are given below:

$$MDL_{opt} = min_p[log(E) + \frac{p * log(N)}{N}] \tag{12}$$

$$AIC_{opt} = min_p[log(E) + 2p/N] \tag{13}$$

E = loss function (e.g. MSE)
p = model order
N = available data points

In Figure 8a, both the MDL and AIC criteria show an optimal prediction for this signal of AR(5). The MDL model appears to better isolate the minima at this point compared to the AIC model. This difference from the actual signal is due to the small signal size. In Figure 8b, the optimal model is defined as AR(4) as expected.

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

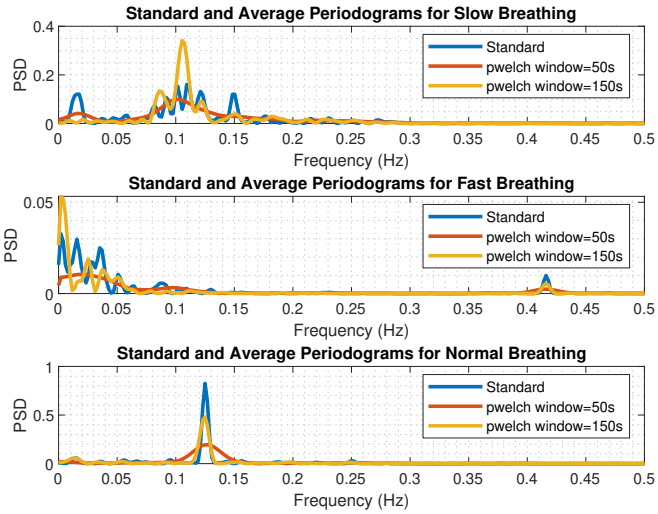


Figure 10: Periodograms of RRI data

Figure 10 shows the standard and averaged periodograms of the RRI data recorded. It can be observed that as the window length shortens, the spectrum becomes smoother. This has the result of reducing noise at the cost of frequency resolution. In this case, the window length of 150s appears to give the best results for identifying the peaks corresponding to the breathing frequency. The relationship between the breathing rate in breaths per minute (BPM) and the frequency is given by $BPM = 2 * 60 * f$. Therefore, breathing rates are given as:

	Frequency Peak Observed (Hz)	**Breathing Rate Observed (BPM)**
Slow Breathing	0.10547	13
Normal Breathing	0.125	15
Fast Breathing	0.4160	50

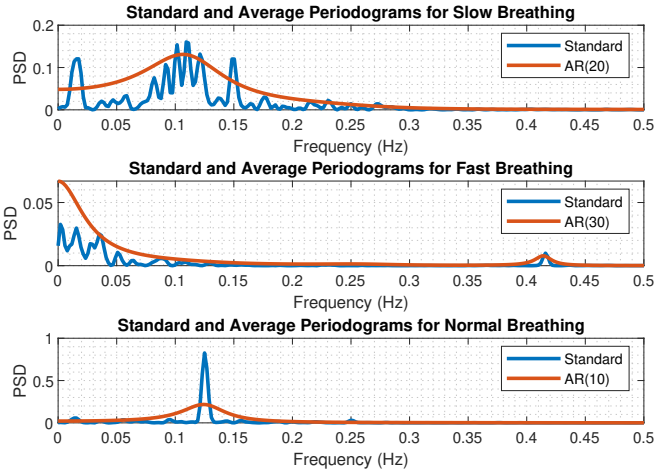


Figure 11: AR models of RRI data

Figure 11 shows the spectra of different AR estimates for the RRI data. A well fitted model will have a defined peak at the relevant breathing frequency. Models AR(20), AR(30) and AR(10) appear to fit the slow, fast and normal breathing RRI data appropriately. Whilst the AR models appear to effectively remove noise, once again this is at the cost of reduced frequency resolution.

1.6 Robust Regression

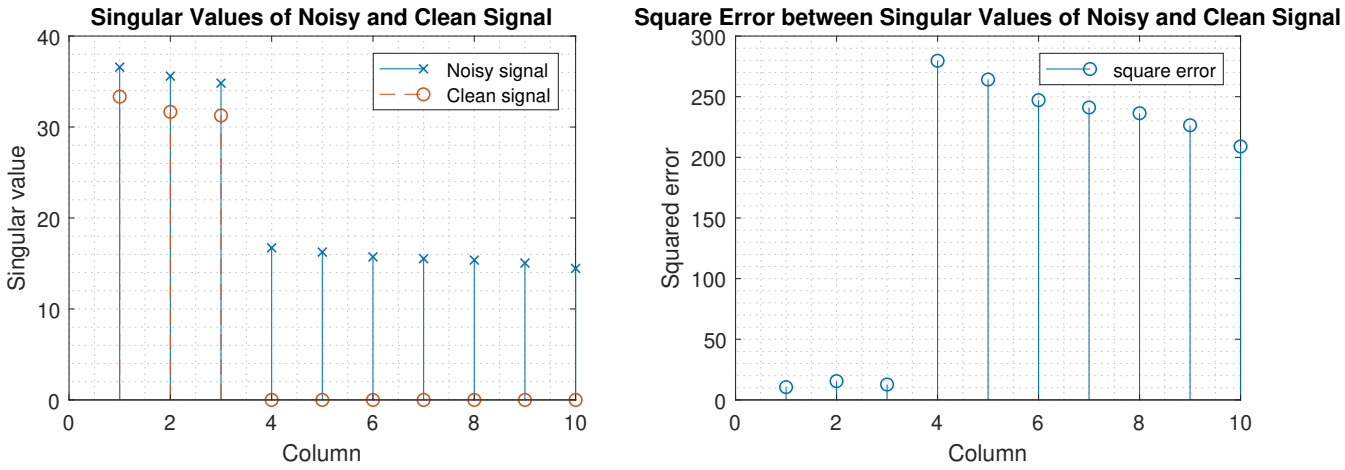


Figure 12: Comparison of singular values between noisy and clean signals

From Figure 12, the rank of the matrix can easily be determined. There are clearly 3 singular values of the clean signal, implying 3 linearly independent columns. Therefore, the input signal $\boldsymbol{X}$ has a rank of 3 (column rank = row rank = overall rank). For the noisy signal; however, there are 10 distinguishable singular values, giving the noisy signal $\boldsymbol{X_{noise}}$ a rank of 10. The reduced row echelon form of the transpose of the signal shows non-zero components in the first 3 rows whilst the other dimensions can be deemed irrelevant. However, noise occupies all 10 dimensions (columns) so the noisy signal is full row rank. In this case, the SNR is relatively large since for the first 3 columns where the main signal components reside, the square error between the clean and noisy signals is very small. This makes it relatively easy to identify the rank of the desired signal in the noisy data. However, if the SNR is small, then the noise becomes a much larger part of the noisy signal and the square error will be similar across all dimensions. Therefore, it would be much more difficult to identify the rank of the signal in $\boldsymbol{X_{noise}}$.

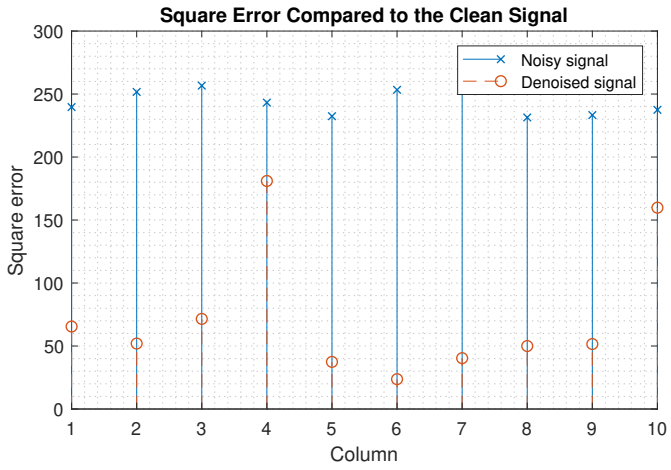


Figure 13: Comparison of noisy vs denoised signals against the clean signal

Figure 13 shows the square error between the clean signal and both the noisy and denoised signals. The denoised signal, $\boldsymbol{X_{denoised}}$, is a low rank approximation of $\boldsymbol{X_{noise}}$ using the 3 most significant components specified by the rank of X. It is clear to see that across every dimension $\boldsymbol{X_{denoised}}$ is a better approximation of $\boldsymbol{X}$ than $\boldsymbol{X_{noise}}$. However, the standard deviation of the error between $\boldsymbol{X_{denoised}}$ and $\boldsymbol{X}$ is also clearly larger than that of the error between $\boldsymbol{X_{noise}}$ and $\boldsymbol{X}$. This technique is referenced as a Principle Component Analysis (PCA).

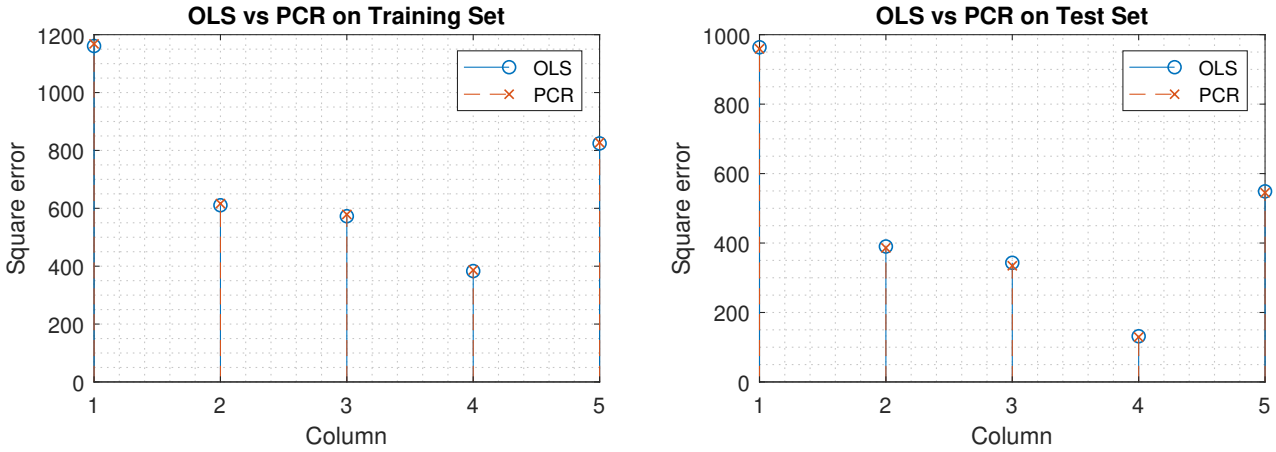


Figure 14: Comparison of OLS and PCR methods

Ordinary Least Squares (OLS) and Principle Component Regression (PCR) are two different regression methods. Both estimates are given below:

$$\boldsymbol{B_{OLS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} \tag{14}$$

$$\boldsymbol{B_{PCR}} = \boldsymbol{V}_{1:r}(\textstyle\sum_{1:r})^{-1}\boldsymbol{U}_{1:r}^T\boldsymbol{Y} \tag{15}$$

Figure 14 compares both these methods on a training and test set. The training set is used to fit the regression matrix and the test set is used to ascertain its performance by seeing how the model performs on a new input. It can be observed that the performance of both metrics in this case are very similar. However, taking the sum of the errors on the test set shows a slight preference to the PCR method with a total error given by $error_{pcr} = 2353.9$ which is smaller than the total error of the OCR method $error_{ocr} = 2377.4$.

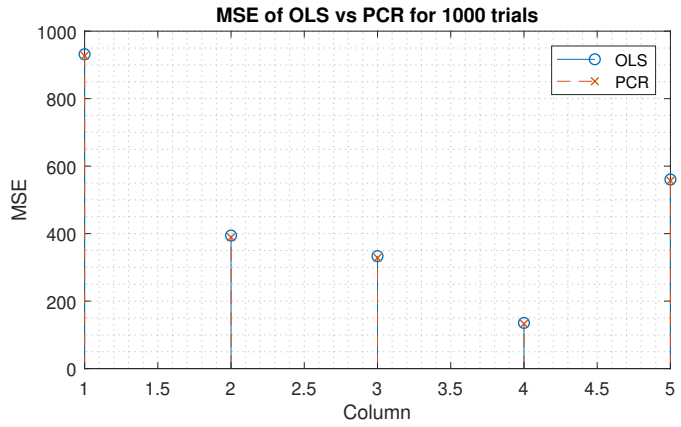


Figure 15: Comparison of OLS and PCR methods (multiple trials)

By having a larger test set, a better model accuracy can be obtained. From Figure 15, both methods again appear to perform very similarly. Once again, if the sum of errors is taken, the PCR method has a slight edge with $error_{pcr} = 2337.0$ compared the OLS method with $error_{ocr} = 2358.0$. In conclusion, the PCR method appears to give a 0.89% improvement in performance. This estimate can be improved by performing more trials and can vary based on the loss function used (e.g. Huber).

2 Adaptive Signal Processing

2.1 The Least Mean Square (LMS) Algorithm

The LMS algorithm is a specialization of gradient descent that uses a mean-squared error function, $J(n) = \frac{1}{2}e^2(n)$. The estimation of the model parameters is given by the weights $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e(n)\boldsymbol{x}(n)$ where μ is the learning rate. The estimate of the signal itself is then given by $x(n)_{LMS} = \boldsymbol{w}^T(n)\boldsymbol{x}(n)$. The instantaneous estimations are important as it allows the model to adapt to a non-stationary signal with model parameters dependent on time. In this section, LMS algorithm will be applied to the AR(2) process given below:

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n) \tag{16}$$

Where, $\eta \sim \mathbb{N}(0, \sigma_\eta^2)$. This process on its own is stationary, so the ACF can be defined purely by the time lag k:

$$\begin{aligned} r_{xx}(k) &= \mathbb{E}\{x(n)x(n-k)\} \\ &= \mathbb{E}\{[a_1 x(n-1) + a_2 x(n-2) + \eta(n)]x(n-k)\} \\ &= a_1\mathbb{E}\{x(n-1)x(n-k)\} + a_2\mathbb{E}\{x(n-2)x(n-k)\} + \mathbb{E}\{\eta(n)x(n-k)\} \end{aligned} \tag{17}$$

The last term here is the cross correlation between the input noise and the output.

$$H(z) = \frac{1}{1 - a_1 z^{-1} - a_2 z^{-2}} \tag{18}$$

$$h(k) = \frac{2^{-k-1}\left(\left(a_1 + \sqrt{a_1^2 + 4a_2}\right)^{k+1} - \left(a_1 - \sqrt{a_1^2 + 4a_2}\right)^{k+1}\right)}{\sqrt{a_1^2 + 4a_2}} \tag{19}$$

$$\begin{aligned} \mathbb{E}\{\eta(n)x(n-k)\} &= R_{\eta\eta}(k) * h^*(-k) \\ &= \sigma_\eta^2\delta(k) * \frac{2^{k-1}\left(\left(a_1 + \sqrt{a_1^2 + 4a_2}\right)^{-k+1} - \left(a_1 - \sqrt{a_1^2 + 4a_2}\right)^{-k+1}\right)\Big)}{\sqrt{a_1^2 + 4a_2}} \\ &= \sigma_\eta^2 \frac{2^{k-1}\left(\left(a_1 + \sqrt{a_1^2 + 4a_2}\right)^{-k+1} - \left(a_1 - \sqrt{a_1^2 + 4a_2}\right)^{-k+1}\right)\Big)}{\sqrt{a_1^2 + 4a_2}} \\ &= \begin{cases} \sigma_\eta^2, & \text{for } k = 0 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{20}$$

This gives three simultaneous equations to solve to obtain solutions $[r_{xx}(0), r_{xx}(1), r_{xx}(2)] = [\frac{25}{27}, \frac{25}{54}, \frac{85}{108}]$. Acknowledging $a_1 = 0.1$, $a_2 = 0.8$ and $\sigma_\eta^2 = 0.25$, the correlation matrix, $\boldsymbol{R_{xx}}$ of the input vector $\boldsymbol{x}(n) = [x(n-1), x(n-2)^T]$ is given by:

$$\begin{aligned} \boldsymbol{R_{xx}} &= \begin{pmatrix} r_{xx}(n-1, n-1) & r_{xx}(n-1, n-2) \\ r_{xx}(n-2, n-1) & r_{xx}(n-2, n-2) \end{pmatrix} \\ &= \begin{pmatrix} r_{xx}(0) & r_{xx}(1) \\ r_{xx}(1) & r_{xx}(0) \end{pmatrix} \\ &= \begin{pmatrix} a_1 r_{xx}(1) + a_2 r_{xx}(2) + \sigma_\eta^2 & a_1 r_{xx}(0) + a_2 r_{xx}(1) \\ a_1 r_{xx}(0) + a_2 r_{xx}(1) & a_1 r_{xx}(1) + a_2 r_{xx}(2) + \sigma_\eta^2 \end{pmatrix} \\ &= \frac{25}{54}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \end{aligned} \tag{21}$$

Eigenvalue decomposition of the autocorrelation matrix gives $\lambda_1 = 0.463$ and $\lambda_2 = 1.389$. Furthermore, the LMS algorithm converges in the mean if step size μ satisfies:

$$0 < \mu < \frac{2}{\lambda_{max}}$$
$$0 < \mu < 1.44 \tag{22}$$

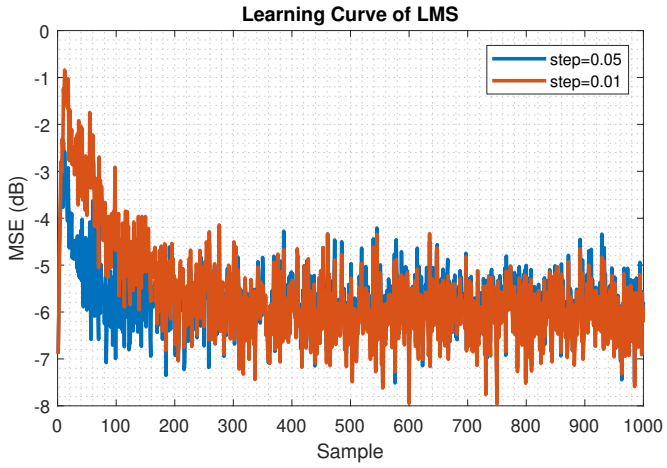


Figure 16: LMS learning curve of AR(2) process for different step sizes

Figure 16 illustrates the learning curves for an LMS predictor of (16) for different step sizes, $[\mu_1, \mu_2] = [0.01, 0.05]$. Clearly, the larger step size $\mu_2 = 0.05$ leads to a faster convergence of the error. Furthermore, the steady state error of the smaller step size $\mu_1 = 0.01$ appears to be slightly smaller. These properties can be explained by the fact that the LMS is a gradient descent algorithm. Therefore, a larger step size leads to a larger step down the curve towards the minima of the error function so convergence is faster. However, at steady state, the algorithm will oscillate around the minimum point and the deviation away from this point is also proportional to the step size. Furthermore, if the step size is too large, the algorithm will overshoot and can diverge away from the local minima as defined by equation (22).

The Excess Mean Square Error (EMSE) measures the difference between the mean-square error introduced by the adaptive filters and the minimum attainable mean square error of the Wiener filter. The misadjustment is the ratio of the excess mean square error and the minimum mean square error, $M = \frac{MSE}{\sigma_w^2} - 1$. Furthermore, an approximation of the misadjustment is given by $M_{LMS} \approx \frac{\mu}{2} Tr(\boldsymbol{R})$.
The LMS is applied to the signal $x(n)$ for 100 trials to get some numbers for the EMSE and misadjustment at different step sizes.

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

The smaller error for the smaller step size $\mu_1 = 0.01$ makes sense since the approximation for misadjustment is derived assuming small step sizes.

The time constant of the error indicates how long the error takes to fall by a factor of e^{-1} of its initial value. The equation for the LMS is given below:

$$\tau_k = -\frac{1}{\ln(|1 - 2\mu\lambda_k|)} \tag{23}$$

λ_k = Smallest eigenvalue (corresponds to the slowest mode)

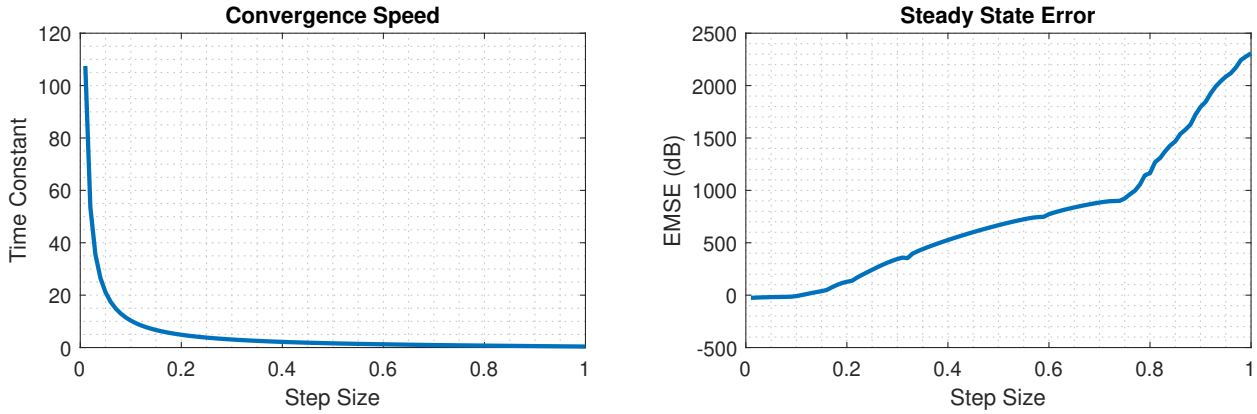


Figure 17: Steady state error and convergence speed relationships

Figure 17 shows how the convergence speed and the steady state EMSE varies with step size for the AR(2) process defined in (16). Here, a step size of $\mu* = 0.1$ appears to give a good trade-off between convergence time and steady state error. Although of course this will depend on the requirements of the specific situation.

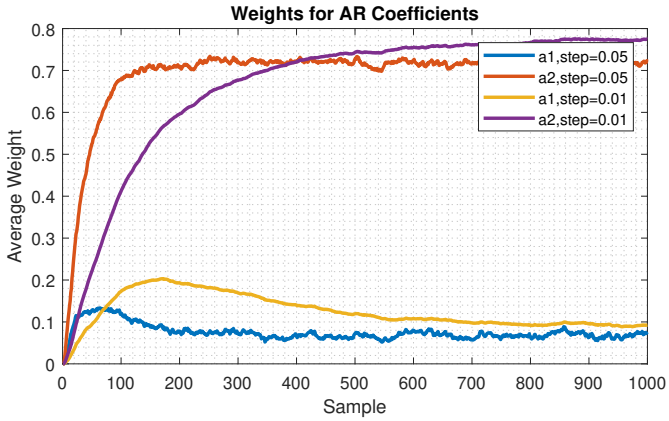


Figure 18: LMS weight evolution for AR(2) process at different step sizes

Figure 18 illustrates how the average weights of the LMS algorithm updates over 100 trials. It is important to remember the values for $a_1 = 0.1$ and $a_2 = 0.8$. It can again be observed that the larger step size of 0.05 leads to a faster rate of convergence at the cost of a larger steady state error. It should also be noted that the transient response of the estimates for a_1 in both cases overshoot substantially before reaching steady state.

Given some leakage coefficient $0 < \gamma < 1$, the cost function $J(n) = \frac{1}{2}(e^2(n) + \gamma||\boldsymbol{w}(n)||_2^2)$ is defined. What follows is the LMS weight update algorithm that minimizes this cost function:

$$J(n) = \frac{1}{2}[(x(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n))^T(x(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n)) + \gamma\boldsymbol{w}(n)^T\boldsymbol{w}(n)] \tag{24}$$

$$\begin{aligned}\frac{d[J(n)]}{d[\boldsymbol{w}(n)]} &= -(x(n) - \boldsymbol{w}(n)^T\boldsymbol{x}(n))\boldsymbol{x}(n) + \gamma\boldsymbol{w}(n) \\ &= -e(n)\boldsymbol{x}(n) + \gamma\boldsymbol{w}(n)\end{aligned} \tag{25}$$

$$\begin{aligned}\boldsymbol{w}(n+1) &= \boldsymbol{w}(m) + \mu\left(-\frac{d[J(n)]}{d[\boldsymbol{w}(n)]}\right) \\ &= \boldsymbol{w}(n) + \mu(e(n)\boldsymbol{x}(n) - \gamma\boldsymbol{w}(n)) \\ &= (1 - \mu\gamma)\boldsymbol{w}(n) + \mu e(n)\boldsymbol{x}(n)\end{aligned} \tag{26}$$

This final expression is the equation for the leaky LMS algorithm.

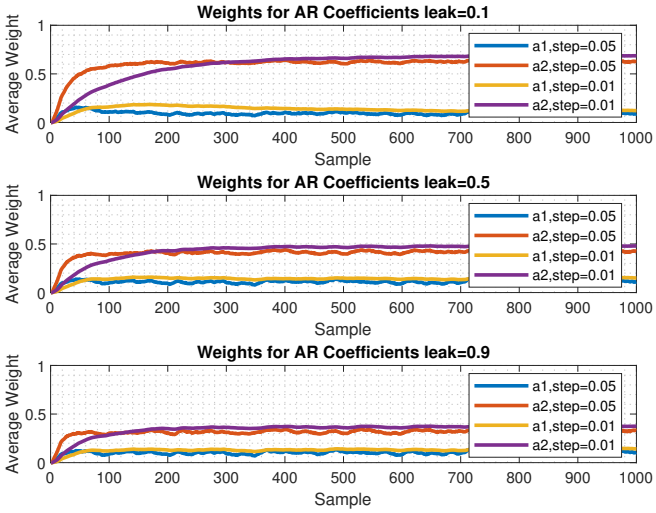


Figure 19: LMS weight evolution for AR(2) process at different step sizes and leak coefficients

Figure 19 illustrates the leaky LMS weight estimation for the same AR(2) process $x(n)$ with different step sizes and leak coefficients. As the leak coefficient increases, the steady state becomes less smooth and the weights converge further and further away from the correct values. The non-adaptive but optimal solution to this linear system is the Wiener filter, which has access to all the observed samples and relies on the autocorrelation matrix, $R = \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^t]$ and the cross-correlation vector between the ideal output and the input, $\boldsymbol{p}$. The Wiener filter weights are given by:

$$\boldsymbol{w}^* = \boldsymbol{R}^{-1}\boldsymbol{p} \tag{27}$$

The key is that $\boldsymbol{R}$ must be inveritble for a solution to exist. The LMS algorithm can converge to this solution if the step size is bounded as in (22). However, invertibility of the autocorrelation is not guaranteed since it is only positive semi-definite. The Leaky LSM algorithm ensures the inveritbility of this by modifying the autocorrelation matrix to $(\boldsymbol{R} + \gamma\boldsymbol{I})$:

$$\boldsymbol{w}^{leaky} = (\boldsymbol{R} + \gamma\boldsymbol{I})^{-1}\boldsymbol{p} \tag{28}$$

However, by ensuring the invertibility of the autocorrelation matrix, a bias has been introduced.

2.2 Adaptive Step Sizes

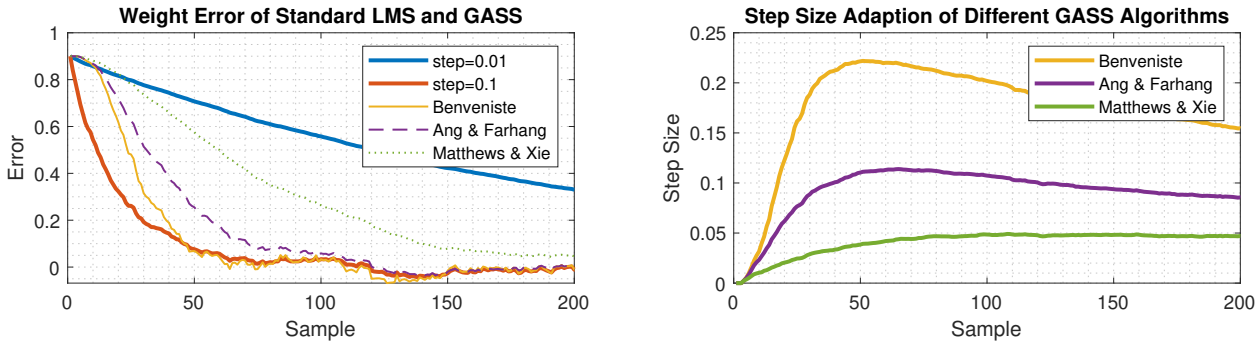
As seen in Section 2.1, there is trade-off when choosing a step size between convergence speed and steady state error variance. Ideally, the step size should begin large and then reduce as the steady state is reached. The Gradient Adaptive Step Size (GASS) algorithms aim to update the learning rate as below:

$$\mu(n+1) = \mu(n) + \rho e(n)\boldsymbol{x}^T(n)\boldsymbol{\psi}(n) \tag{29}$$

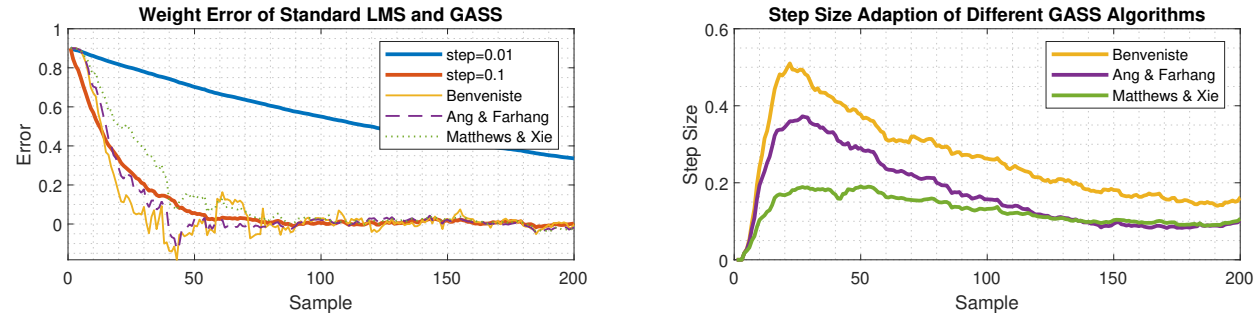
Where $\boldsymbol{\psi}(n)$ is given by one of Benveniste, Ang Farhang or Matthews Xie. This will be applied on a MA(1) process given below:

$$x(n) = 0.9\eta(n-1) + \eta(n) \tag{30}$$

Where, $\eta \sim \mathbb{N}(0, 0.5)$.

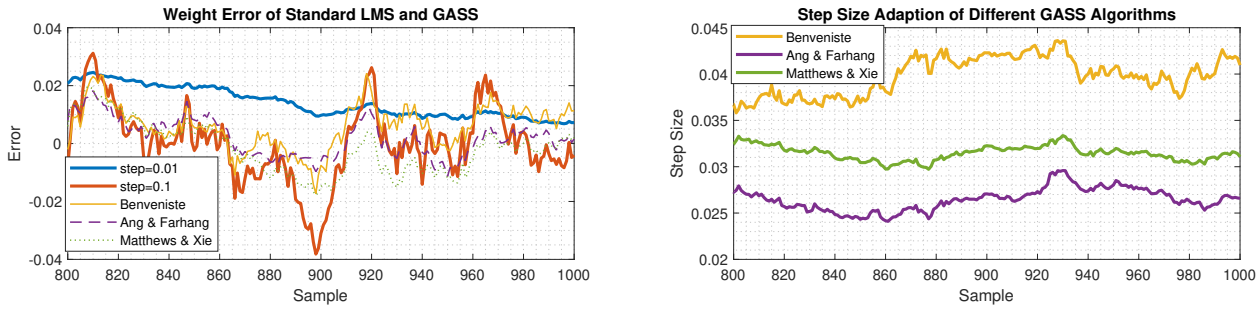


(a) $\rho = 0.005$

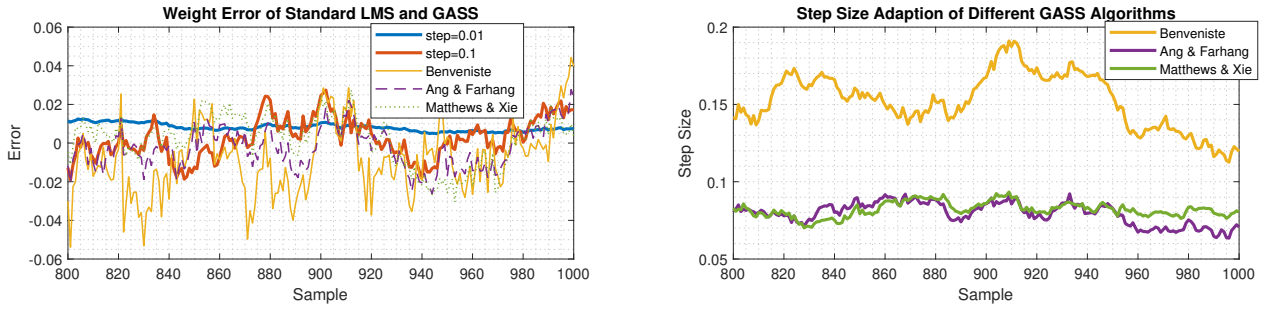


(b) $\rho = 0.05$

Figure 20: Transient response of GASS and standard LMS algorithms



(a) $\rho = 0.005$



(b) $\rho = 0.05$

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

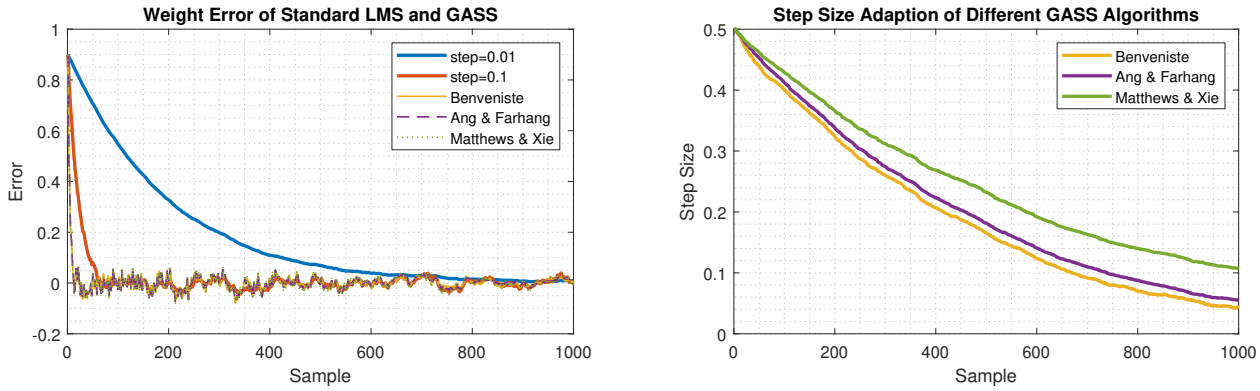


Figure 22: Overall response of GASS and standard LMS algorithms with more optimized GASS parameters

Figure 20 shows how the transient response of the different GASS algorithms with initial step sizes $\mu^0_{GASS} = [\mu^0_B, \mu^0_{AF}, \mu^0_{MX}]$ compares with the standard LMS algorithm with step sizes $[\mu_1, \mu_2] = [0.01, 0.1]$. As expected, the standard LMS algorithm with step size μ_1 converges the slowest whilst the algorithm with step size μ_2 appears to converge fastest. However, it should be observed that the initial step size of the GASS algorithms is given by $\mu^0_{GASS} = 0$, and the Ang Farhang and Benveniste algorithms have $max(\mu_{AF}) > 0.1, max(\mu_B) > 0.1$ before reaching steady state. Therefore, by choosing an appropriate initial step size, the convergence time of the GASS algorithms should greatly improve. Furthermore, by choosing a larger ρ value, the step size of the GASS algorithms climb much faster and so converge faster as well. It can be observed that the Benveniste algorithm converges fastest out of all the GASS algorithms whilst the Matthews Xie algorithm converges the slowest. This is because the step size of the Benveniste algorithm climbs fastest and vice versa for the Matthews Xie algorithm.
Figure 21 shows how the steady state response of the different GASS algorithms compares with the standard LMS algorithm. The standard LMS algorithm with step size μ_1 clearly has the least steady state variance. However, in Figure 21a, where ρ is small, all three GASS algorithms perform better in the steady state than the LMS algorithm with step size μ_2. However, in Figure 21b, where ρ is large, the Benveniste algorithm clearly performs worst. Once again, the Benveniste algorithm has the largest steady state step size and the Ang Farhang algorithm appears to settle at the lowest step size across both ρ values.
Clearly, the value of ρ and the initial step size can greatly impact the performance of the GASS algorithms. By optimizing both these values as in Figure 22 to $\rho = 0.005$ and $\mu^0_{GASS} = [0.5, 0.5, 0.5]$, the Benveniste algorithm adapts best with the steepest gradient descent and smallest overall step size. This is expected as it has the largest time complexity of $O(N^2)$ where N is the model order. In the steady state, $[EMSE_B, EMSE_{AF}, EMSE_{MX}] = [-3.08dB, -3.05dB, -2.99dB]$ whilst the standard LMS has steady state performance, $[EMSE_{\mu=0.01}, EMSE_{\mu=0.1}] = [-3.13dB, -3.02]$. The learning curves for the LMS algorithms and optimized GASS algorithms are shown in Figure 23.

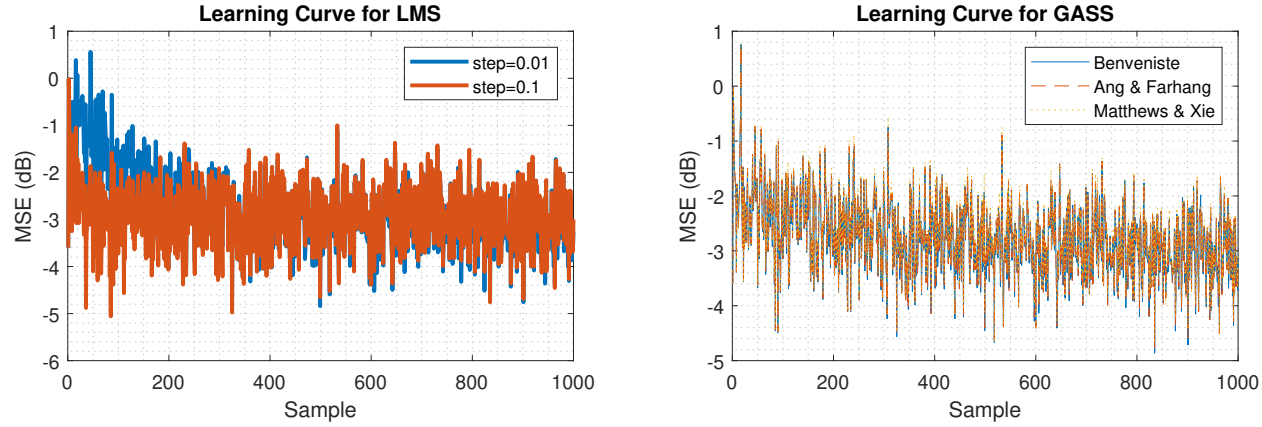


Figure 23: Learning Curves of LMS and GASS algorithms

It is often useful to normalise the step size using the normalized LMS (NLMS) algorithm. Given the a posteriori error $e_p(n) = d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n+1)$ and the a priori error $e(n) = d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n)$, start with the standard update equation:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e_p(n)\boldsymbol{x}(n) \tag{31}$$

Modify this equation to allow the substitution of the a priori and a posterirori error functions:

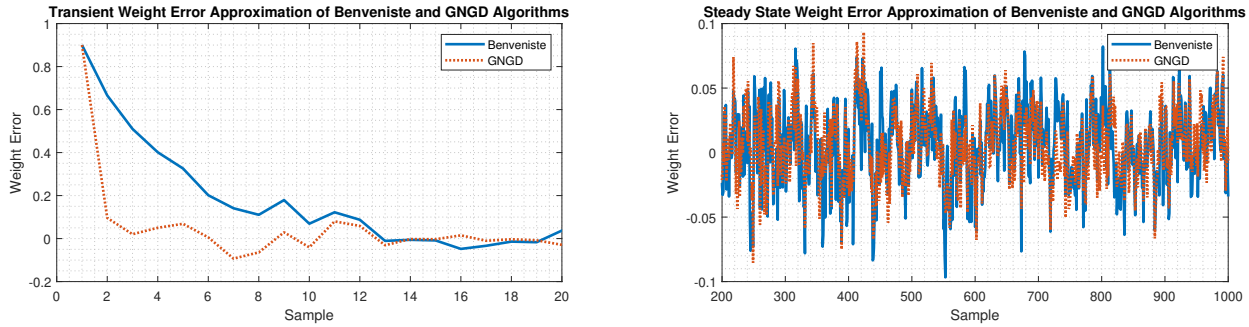
$$d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n+1) = d(n) - \boldsymbol{x}^T(n)\boldsymbol{w}(n) - \boldsymbol{x}^T(n)\mu e_p(n)\boldsymbol{x}(n) \tag{32}$$

$$\begin{aligned} e_p(n) &= e(n) - \mu e_p(n)\boldsymbol{x}^T(n)\boldsymbol{x}(n) \\ &= \frac{e(n)}{1+\mu\boldsymbol{x}^T(n)\boldsymbol{x}(n)} \end{aligned} \tag{33}$$

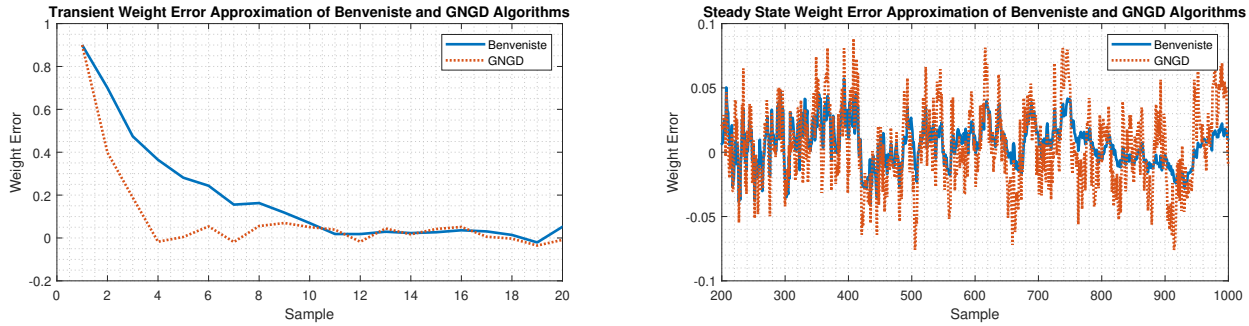
Now, substituting this equation for the a posteriori error to the update equation:

$$\begin{aligned} \boldsymbol{w}(n+1) &= \boldsymbol{w}(n) + \frac{\mu}{1+\mu\boldsymbol{x}^T(n)\boldsymbol{x}(n)}e(n)\boldsymbol{x}(n) \\ &= \boldsymbol{w}(n) + \frac{1}{\frac{1}{\mu}+\boldsymbol{x}^T(n)\boldsymbol{x}(n)}e(n)\boldsymbol{x}(n) \\ &= \boldsymbol{w}(n) + \frac{\beta}{\varepsilon+\boldsymbol{x}^T(n)\boldsymbol{x}(n)}e(n)\boldsymbol{x}(n) \end{aligned} \tag{34}$$

This is the NLMS update equation where $\beta = 1$ and $\varepsilon = \frac{1}{\mu}$. Therefore, the NLMS algorithm is the standard LMS algorithm based on the a posteriori error. The gain of the NLMS algorithm can be made adaptive by making the regularization factor ε time varying. This can be done using the generalized normalized gradient descent algorithm (GNGD).



(a) $\rho = 0.0005$, $\mu^0_{GNGD} = 7$, $\mu^0_{GASS} = 0.5$

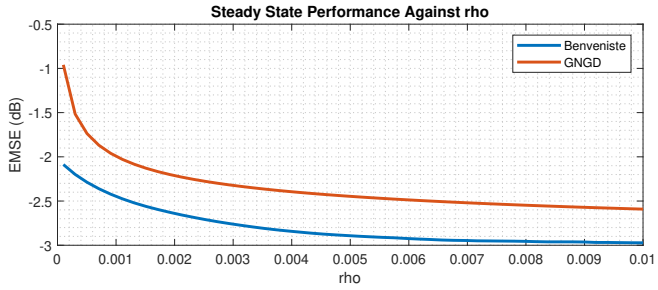


(b) $\rho = 0.005$, $\mu^0_{GNGD} = 7$, $\mu^0_{GASS} = 0.5$

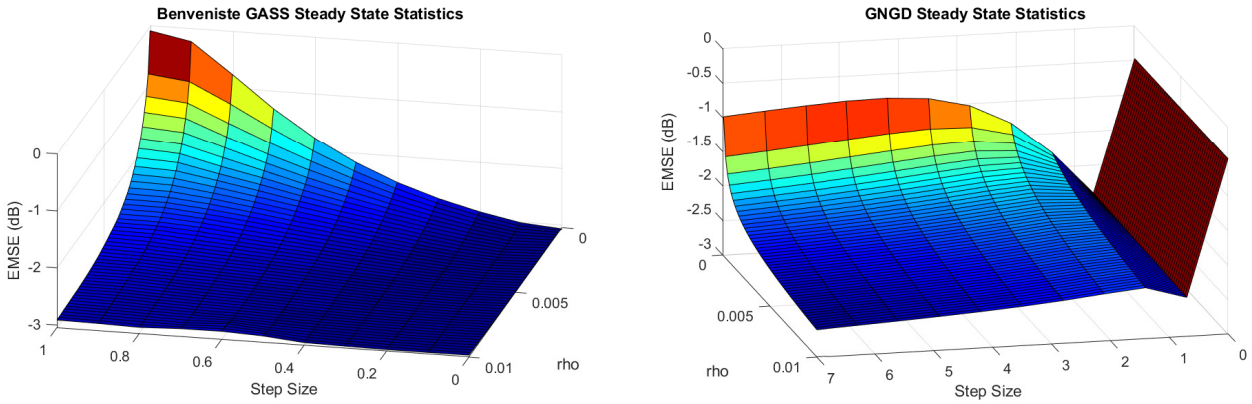
Figure 24: Transient and steady state response of Benveniste GASS and GNGD algorithms

Figure 24 compares the Benveniste GASS algorithm with the GNGD algorithm. The key aspect to note about the GNGD algorithm is that it is very robust to large step sizes whereas the GASS algorithm is not. Therefore, it can be made to converge much faster than the GASS Benveniste algorithm by choosing a very large step size; as can be seen in Figure 24. The ρ value greatly affects the steady state performance of both algorithms. For small $\rho = 0.0005$ in Figure 24a, the GASS has a smaller EMSE when compared to the GNGD algorithm, $[EMSE^{\rho=0.0005}_{GNGD}, EMSE^{\rho=0.0005}_{GASS}] = [-1.74dB, -2.29dB]$. For large $\rho = 0.005$ in Figure 24b, the EMSE of the GASS algorithm greatly improves compared to the GNGD algorithm, $[EMSE^{\rho=0.005}_{GNGD}, EMSE^{\rho=0.005}_{GASS}] = [-2.45dB, -2.89dB]$. For the same change in ρ, the Benveniste algorithm improves by 26.2% whereas the GNGD algorithm imporves by 40.8%. This relationship with ρ is examined further in Figure 25.
The GNGD is also less computationally complex than the GASS algorithm; where the GASS algorithm has complexity $O(N^2)$, the GNGD algorithm has complexity $O(N)$. This is because each update of the GNGD algorithm involves only inner product calculations and additions of N-dimensional vectors and scalars.

In conclusion, Figure 25 illustrates how the steady state performance of both the Benveniste GASS and GNGD algorithms compare. The GNGD algorithm has the best steady state performance with an initial step size $\mu^0_{GNGD} = 7$ and $\rho = 0.01$ giving $EMSE^*_{GNGD} = 0.5506 = -2.59dB$. The Benveniste GASS algorithm has the best steady state performance with an initial step size $\mu^0_{GASS} = 0.2$ and $\rho = 0.0036$ giving $EMSE^*_{GASS} = 0.4961 = -3.04dB$. However, especially for the GASS Benveniste algorithm, optimal parameter decisions must take into account the convergence time, which is also dependent on both μ^0_{GASS} and ρ.



(a) Steady state performance with $\mu^0_{GNGD} = 7$ and $\mu^0_{GASS} = 0.5$



(b) Steady state performance of algorithms with respect to ρ and μ_0

Figure 25

2.3 Adaptive Noise Cancellation

In this section, different adaptive noise cancellation techniques will be investigated. The noise corrupted signal (35) will be used. The underlying signal $x(n)$ has frequency $f_x = 0.005Hz$ and the noise is given by $\eta(n) = v(n) + 0.5v(n-2)$ where $v(n)$ is white noise with unit variance.

$$s(n) = x(n) + \eta(n) \tag{35}$$

The Adaptive Line Enhancer (ALE) is a noise reduction algorithm consisting of a delay operator and a linear predictor. The delay is chosen such that the noise in the signal and the linear predictor input are uncorrelated. The MSE can be expressed as:

$$\begin{aligned}\mathbb{E}\{(s(n)-\hat{x})^2\} &= \mathbb{E}\{(x(n)+\eta(n)-\hat{x}(n))^2\} \\ &= \mathbb{E}\{(x(n)-\hat{x}(n))^2\} + \mathbb{E}\{\eta^2(n)\} + 2\mathbb{E}\{(x(n)-\hat{x}(n))\eta(n)\} \\ &= \mathbb{E}\{(x(n)-\hat{x}(n))^2\} + \mathbb{E}\{\eta^2(n)\} + 2\mathbb{E}\{x(n)\eta(n)\} - 2\mathbb{E}\{\hat{x}(n)\eta(n)\}\end{aligned} \tag{36}$$

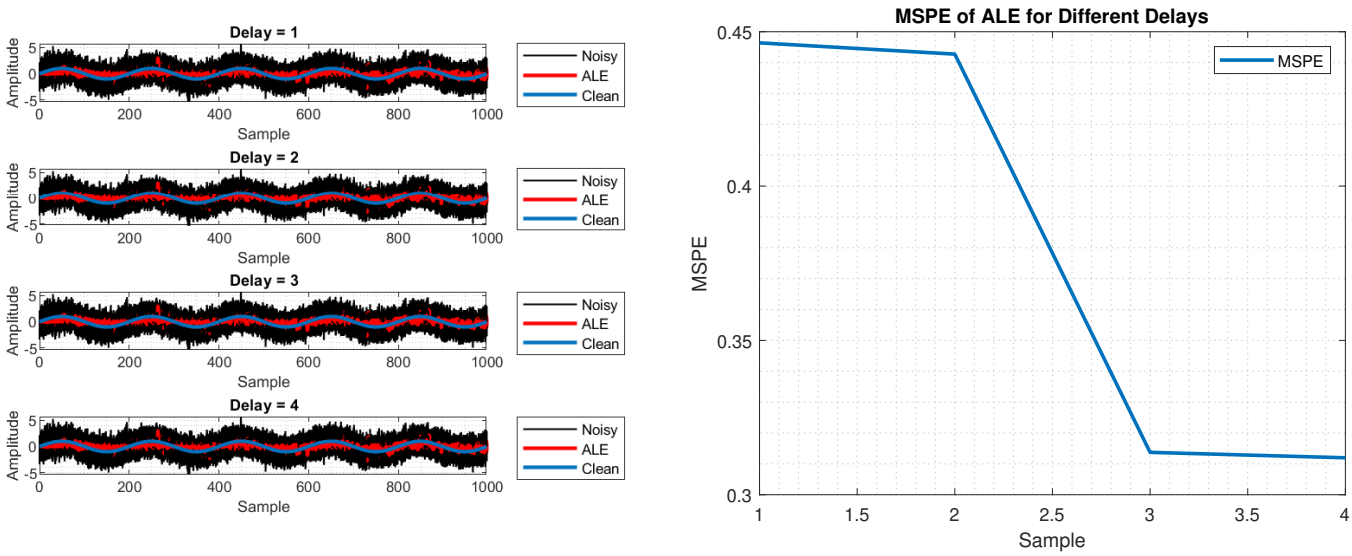
The $\mathbb{E}\{(x(n)-\hat{x}(n))^2\}$ term is not a function of noise. The second term $\mathbb{E}\{\eta^2(n)\}$ is the noise power which is constant. Finally, the third term $2\mathbb{E}\{x(n)\eta(n)\}$ is 0 since the signal and noise are uncorrelated and the noise has zero mean. Therefore, only the last term is considered to minimise the MSE:

$$
\begin{aligned}
\mathbb{E}\{\hat{x}_\Delta(n)\eta(n)\} &= \mathbb{E}\{\boldsymbol{w}^T(n)\boldsymbol{u}_\Delta(n)\eta(n)\} \\
&= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k s(n-\Delta-k)\eta(n)\right\} \\
&= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k (x(n-\Delta-k)+\eta(n-\Delta-k))\eta(n)\right\} \\
&= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k \eta(n-\Delta-k)\eta(n)\right\} \\
&= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k (v(n-\Delta-k)+0.5v(n-\Delta-k-2))(v(n)+0.5v(n-2))\right\}
\end{aligned}
\tag{37}
$$

Since $v(n)$ is identically and independently distributed white noise:

$$
\mathbb{E}\{v(n-i)v(n-j)\} = 0, \text{ for } i \neq j \tag{38}
$$

Therefore, the minimum of (36) is 0 for $\Delta > 2$ so $\Delta_{min} = 3$ to ensure the terms are not time-overlapping. This is expected since $\eta(n)$ is an MA(2) process.



(a) ALE performance at different delays

(b) MSPE at different delays

Figure 26

Figure 26 supports this conclusion. The MSPE is clearly lowest from $\Delta = 3$ onwards where it plateaus.

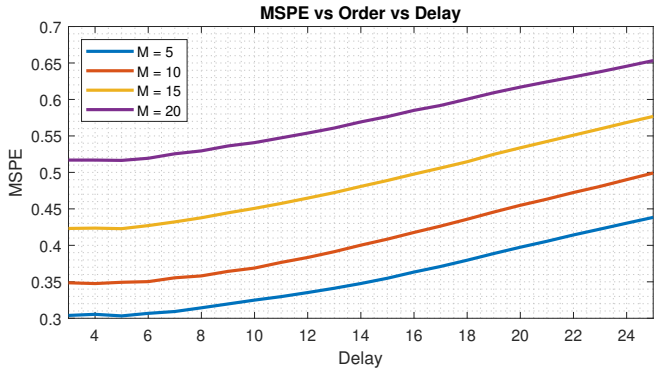


Figure 27: ALE performance based on order and delay

Figure 27 shows the relationship between the ALE MSPE with the delay and model order. It can be observed that the best choice of parameters is $M = 5$ and $\Delta = 3$.
A large model order leads to excess degrees of freedom that increase computational complexity and reduce the model performance. This is because the model over-fits the noise as part of the signal.
for $\Delta > 5$, the ALE model also begins to deteriorate with increasing MSPE. This is because the first term of (36) begins to have a bigger impact where there is a time shift between the filter output and the clean signal. Therefore, as the delay approaches the period of the signal, the error comes back down, following a periodic pattern.

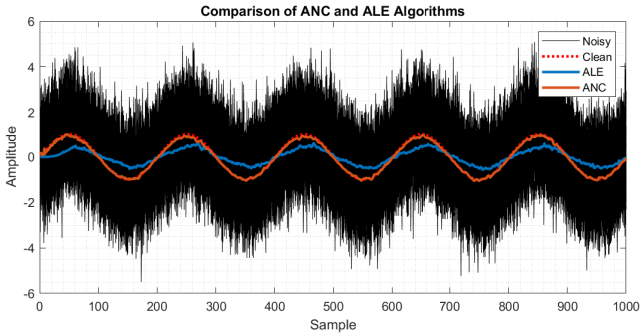
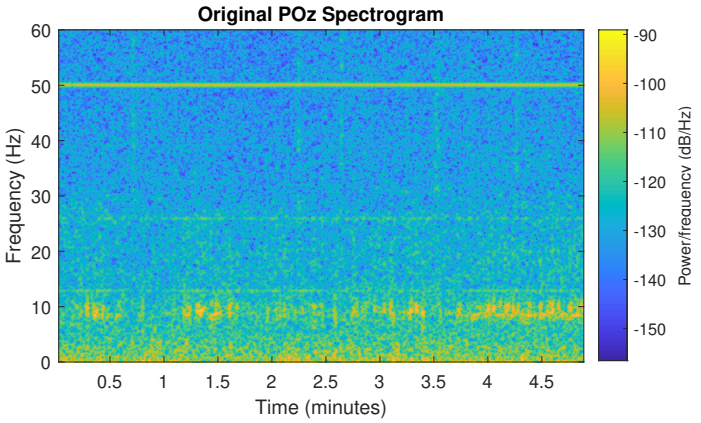


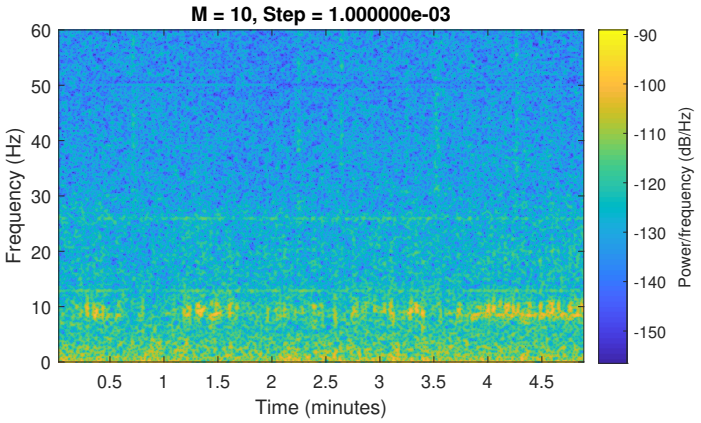
Figure 28: ALE vs ANC algorithms

The ANC algorithm works by taking in a noisy signal (35) as well as a secondary noise input which is correlated in some unknown way to the primary noise. This secondary noise is passed to a linear predictor which aims to obtain the primary noise. This can then be subtracted from the original noisy input to obtain the clean signal. Figure 28 shows the performance of both the ANC and ALE algorithms for the same noisy input signal. It can be observed that the ANC output is much better fitted to the clean signal compared to the ALE output. This is supported by the MSPE values: $MSPE_{ANC} = 0.1084$ and $MSPE_{ALE} = 0.3137$. From this statistic, the ANC algorithm appears to perform 65.4% better than the ALE algorithm.

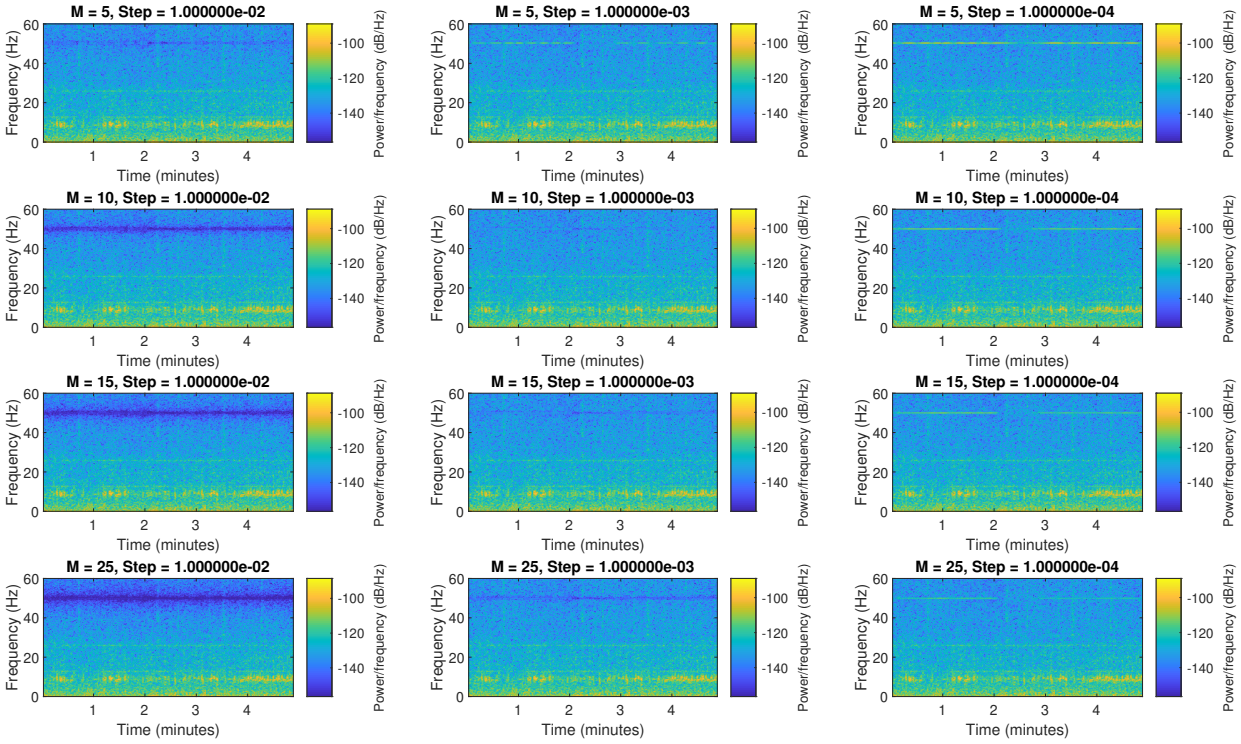
Figure 29 shows how the ANC filter can be used to remove 50Hz mains interference from a real data set. Both the filters with $\mu = 0.001$ and $M = [10, 15]$ appear to best remove the interference whilst maintaining the signal component (maximizing SNR). However, the filter with model order 10 is chosen since it is less computationally complex. This leads to $MSPE = 1.2022 \times 10^{-11}$. Since MSPE measures the error between the original POz signal and the denoised signal, a lower value does not necessarily indicate a better performance in this case. Clearly, the ANC algorithm is very powerful and is often used in the real world; for example, in high end noise cancelling headphones (Bose QC 35, Sony WH-1000XM3, etc).



(a) Original spectrogram



(b) Optimal spectrogram



(c) Spectrograms with different ANC parameters

Figure 29

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

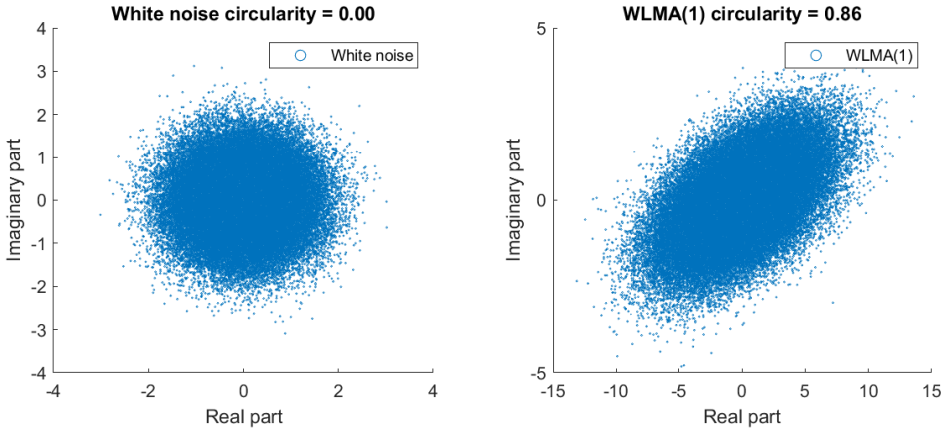
The complex LMS (CLMS) is used to estimate complex data, but only caters for circular data. To better account for general complex data, a widely linear framework is used in the augmented CLMS (ACLMS) given by:

$$\hat{y}(n) = \boldsymbol{h}^H(n)\boldsymbol{x}(n) + \boldsymbol{g}^H(n)\boldsymbol{x}^*(n) \tag{39}$$
$$e(n) = y(n) - \hat{y}(n) \tag{40}$$
$$\boldsymbol{h}(n+1) = \boldsymbol{h}(n) + \mu e^*(n)\boldsymbol{x}(n) \tag{41}$$
$$\boldsymbol{g}(n+1) = \boldsymbol{g}(n) + \mu e^*(n)\boldsymbol{x}^*(n) \tag{42}$$

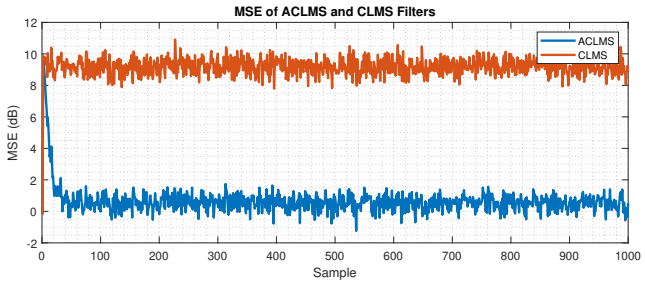
This will be applied to the WLMA(1) signal:

$$y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-1) \tag{43}$$

$x \sim \mathbb{N}(0,1)$
$b_1 = 1.5 + 1j$
$b_2 = 2.5 - 0.5j$



(a) Circularity of circular noise and WLMA signal



(b) Error plot of ACLMS and CLMS filters

Figure 30

Figure 30a shows the rotation of both the circular noise and the WLMA(1) signal $y(n)$. From these results, the expected performance of the ACLMS filter is predicted to be much better than the CLMS filter since the CLMS only caters for circular data. Figure 30b confirms this hypothesis. It is clear to see the steady state MSE for the ACLMS prediction is much lower than for the CLMS prediction. In this case, $EMSE_{ACLMS} = 0.1352$ and $EMSE_{CLMS} = 7.2742$. In fact, the ACLMS outperfors CLMS for noncircular signals whilst having identical performance for circular data. The only advantage the CLMS algorithm has is a faster convergence time since it has half the parameters to train compared to the ACLMS algorithm.

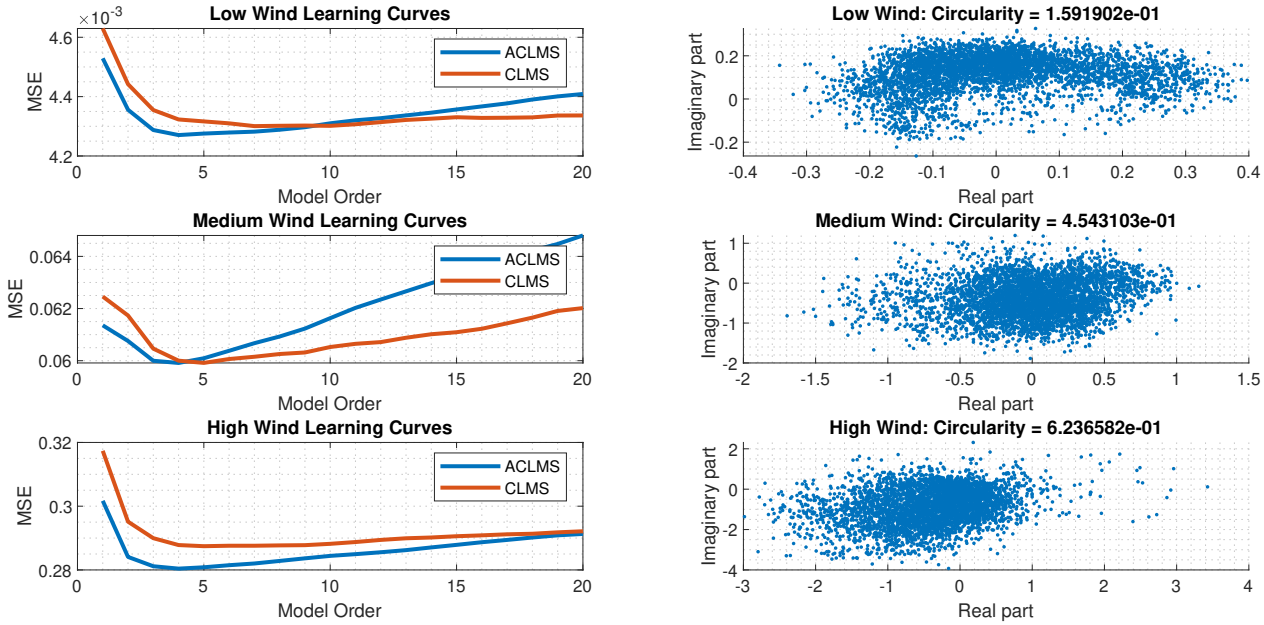


Figure 31: ACLMS and CLMS learning curves for different wind data

Figure 31 shows a comparison between the CLMS and ACLMS filters applied to real wind data. From the circularity plots, it is clear that the low wind data is the most circular whilst the high wind data is the least circular. Furthermore, the optimal step size for both fast convergence and adequate steady state variance required for each data set is different; $[\mu_{low}, \mu_{medium}, \mu_{high}] = [0.1, 0.01, 0.001]$. With these parameters set, it can be observed that in all three cases the ACLMS predictor has a lower minimum MSE when compared to the CLMS predictor. Furthermore, the optimal model order for each set is given by $[M_{low}, M_{medium}, M_{high}] = [4, 4, 4]$. If the ratio between the minimum CLMS filter output and minimum ACLMS filter output is taken, the resultant ratios are given by $[r_{low}, r_{medium}, r_{high}] = [1.0071, 1.0000371, 1.0251]$. From these ratios, it is clear that the ACLMS algorithm outperforms the CLMS algorithm the most for the high wind data which is expected since it is the least circular. However, surprisingly the CLMS algorithm competes best with the ACLMS algorithm for the medium wind data, and not the low wind data which is the most circular.

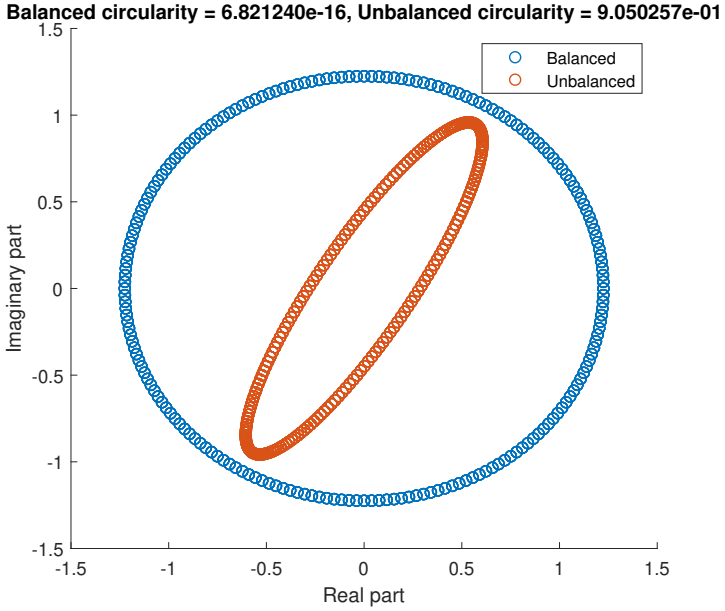


Figure 32: Balanced and unbalanced circularity plots

Figure 32 shows the circularity of both balanced and unbalanced three phase power systems. A balanced system is defined as one where the amplitude of all three phases are the same and the phase shift between each is exactly $\frac{2}{3}\pi$ radians or 120°. When the system is balanced, the circularity is very close to zero, so the complex real-imaginary plot looks circular. In this case, the unbalanced plot has differing phase amplitudes as well as different phase shifts between phases. As a result, the circularity is much closer to 1, and the complex real-imaginary plot is much more elliptical in shape.

The $\alpha - \beta$ complex voltage $v(n) = v_\alpha(n) + jv_\beta(n)$ is obtained by applying the Clarke Matrix to the three phase voltage system matrix. In a balanced system, this is given by:

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n + \phi)} \tag{44}$$

So the voltage at the next time step is:

$$\begin{aligned} v(n+1) &= \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} \\ &= \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_o}{f_s} n + \phi)} e^{j2\pi \frac{f_o}{f_s}} \\ &= v(n) e^{j2\pi \frac{f_o}{f_s}} \end{aligned} \tag{45}$$

Consider the strictly linear autoregressive model:

$$v(n+1) = h^*(n) v(n) \tag{46}$$

Therefore, $h^*(n)$ can be expressed in (45):

$$h^*(n) = e^{j2\pi \frac{f_o}{f_s}} \tag{47}$$

$$\begin{aligned} h(n) &= e^{-j2\pi \frac{f_o}{f_s}} \\ h(n) &= |h(n)| e^{j(\arctan(\frac{\Im(h(n))}{\Re(h(n))}))} \end{aligned} \tag{48}$$

Equating phases:

$$-2\pi \frac{f_o}{f_s} = \arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right) \tag{49}$$

$$f_o = -\frac{f_s}{2\pi} \arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right) \tag{50}$$

For the unbalanced system:

$$v(n) = A(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)} + B(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)} v(n+1) = A(n+1) e^{j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} + B(n+1) e^{-j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} \tag{51}$$

$$\begin{aligned} v(n+1) &= A(n+1) e^{j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} + B(n+1) e^{-j(2\pi \frac{f_o}{f_s}(n+1) + \phi)} \\ &\approx A(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)} e^{j2\pi \frac{f_o}{f_s}} + B(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)} e^{-j2\pi \frac{f_o}{f_s}} \end{aligned} \tag{52}$$

Now, taking the widely linear autoregressive model:

$$\begin{aligned} v(n+1) &= h^*(n) v(n) + g^*(n) v^*(n) \\ &= h^*(n) \left(A(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)} + B(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)} \right) \\ &+ g^*(n) \left(A^*(n) e^{-j(2\pi \frac{f_o}{f_s} n + \phi)} + B^*(n) e^{j(2\pi \frac{f_o}{f_s} n + \phi)} \right) \end{aligned} \tag{53}$$

Equating (52) and (53):

$$A(n)e^{j(2\pi\frac{f_o}{f_s})} = h^*(n)A(n) + g^*(n)B^*(n) \tag{54}$$

$$B(n)e^{-j(2\pi\frac{f_o}{f_s})} = h^*(n)B(n) + g^*(n)A^*(n) \tag{55}$$

Therefore:

$$e^{j(2\pi\frac{f_o}{f_s})} = h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \tag{56}$$

$$e^{-j(2\pi\frac{f_o}{f_s})} = h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \tag{57}$$

Note that (56) is the complex conjugate of (57):

$$h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} = h(n) + g(n)\frac{A(n)}{B^*(n)} \tag{58}$$

$$g^*(n)\left(\frac{B^*(n)}{A(n)}\right)^2 + (h^*(n) - h(n))\left(\frac{B^*(n)}{A(n)}\right) - g(n) = 0 \tag{59}$$

Solving this quadratic:

$$\begin{aligned}\frac{B^*(n)}{A(n)} &= \frac{-(h^*(n) - h(n)) \pm \sqrt{(h^*(n) - h(n))^2 + 4g^*(n)g(n)}}{2g^*(n)} \\ &= \frac{\Im(h(n))j \pm j\sqrt{\Im(h(n))^2 - |g(n)|^2}}{g^*(n)}\end{aligned} \tag{60}$$

Substituting (60) with (56) and (57):

$$\begin{aligned}e^j 2\pi\frac{f_o}{f_s} &= h^*(n) + \Im(h(n))j \pm j\sqrt{\Im(h(n))^2 - |g(n)|^2} \\ &= \Re(h(n)) \pm j\sqrt{\Im(h(n))^2 - |g(n)|^2}\end{aligned} \tag{61}$$

Keeping the + sign solution since $f_s > f_o > 0$ and equating the phase:

$$2\pi\frac{f_o}{f_s} = \arctan\left(\frac{\sqrt{\Im(h(n))^2 - |g(n)|^2}}{\Re(h(n))}\right) \tag{62}$$

$$f_o = \frac{f_s}{2\pi}\arctan\left(\frac{\sqrt{\Im(h(n))^2 - |g(n)|^2}}{\Re(h(n))}\right) \tag{63}$$

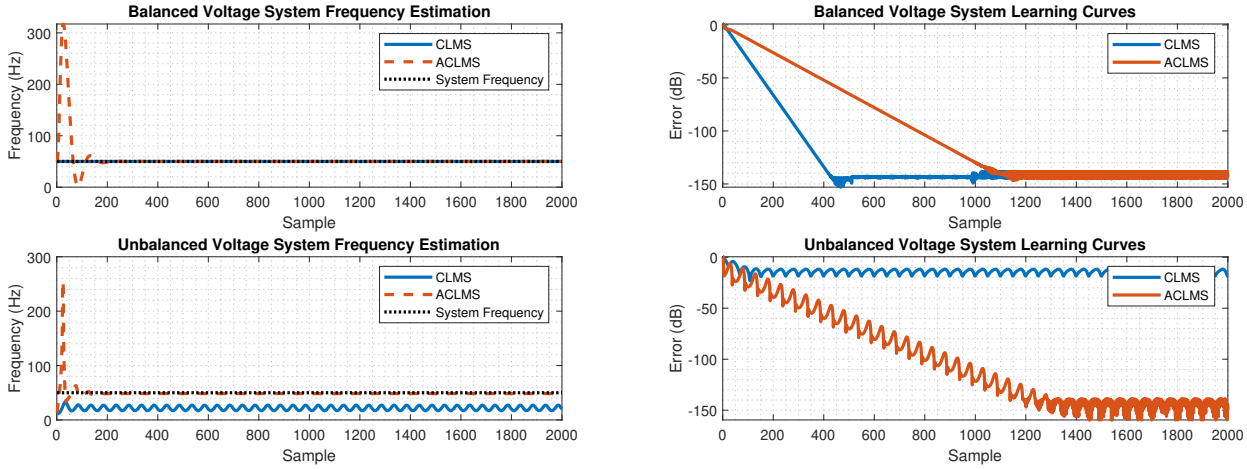


Figure 33: Prediction of three phase voltage system frequency using ACLMS and CLMS filters

Figure 33 shows the nominal frequency estimation of a three phase power system. The system frequency is given by $f_o = 50Hz$ to simulate a UK environment. For the balanced system with circular distribution, both the ACLMS and CLMS filters predict the correct frequency in the steady state. However, the transient response of the ACLMS filter is markedly worse, peaking at 315Hz, and the learning curve of the ACLMS converges slower as well at 1100 samples as opposed to 450 samples.
For the unbalanced system, with non-circular distribution, the ACLMS filter predicts the correct frequency again in the steady state but the CLMS fails to do so with a predicted system frequency of 21Hz. Furthermore, there is a much larger steady state variance with the estimated frequency plot having an amplitude of 5Hz. Once again, there is a large transient response from the ACLMS filter, although this is reduces when compared to the balanced response with a peak at 255Hz. Furthermore, the learning curve converges a little slower at 1300 samples when compared to the balanced response.
These phenomena can be explained by the degrees of freedom in each model. The CLMS algorithm has only one degree of freedom so can only model circular data. Therefore, for the balanced system, it converges almost immediately since the data it is estimating is also circular. However, when it comes to estimating non circular data, the algorithm can only output the nearest circle. The ACLMS algorithm, on the other hand, has an extra degree of freedom. This puts more burden on the model which needs more time to converge with an extra parameter to explore. However, the benefit is that it can also predict elliptical (or non circular) data sets so is able to estimate the unbalanced system in the steady state as well as the balanced system.

3.2 Adaptive AR Model Based Time-Frequency Estimation

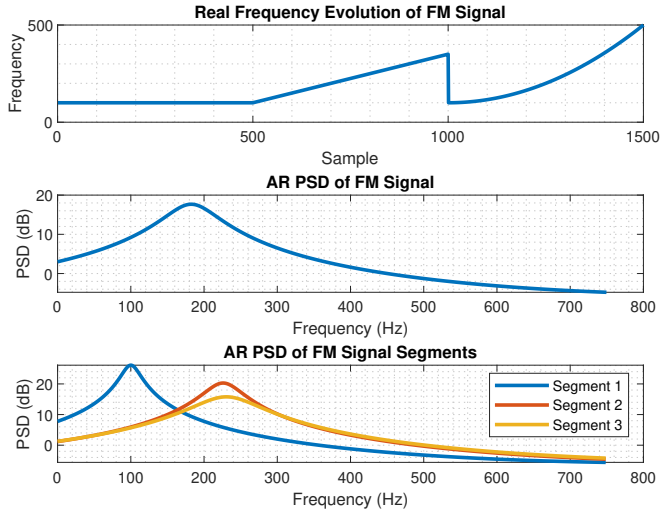


Figure 34: AR frequency estimation of a non-stationary signal

Figure 34 shows the PSD of the AR(1) estimation of a non-stationary FM signal. It can be seen that the signal has three distinct segments, the first having a constant frequency at 100Hz, the second has a linear ramp rising frequency whilst the third has frequency following a quadratic pattern. However, the AR(1) PSD is clearly unable to capture the frequency evolution between segments and only has one clear peak at 185Hz; taking the average of the PSDs of each individual section. Another method is required to appropriately capture the evolution of non-stationary signals.

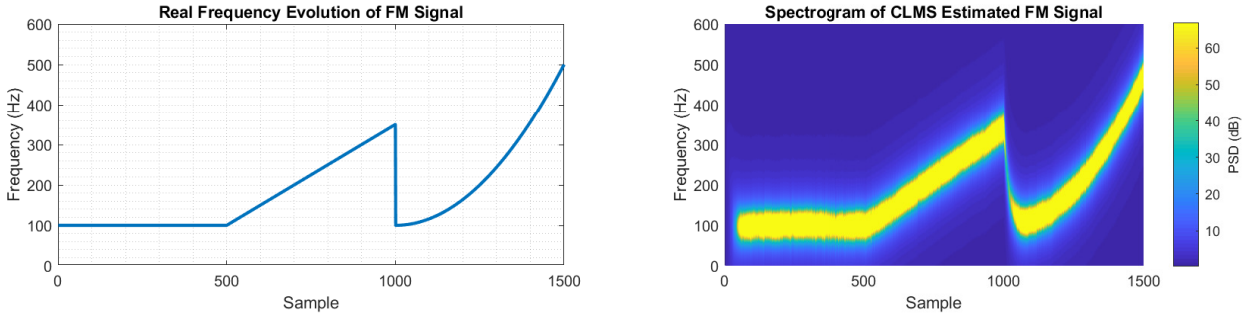


Figure 35: CLMS frequency estimation of a non-stationary signal

Figure 35 shows the spectrogram for the dynamic CLMS estimation of the same non-stationary FM signal, since the signal is complex. The signal itself has a circularity, $c = 0.0098$, so is very circular. This makes the CLMS algorithm a good choice over the ACLMS algorithm for faster convergence and very similar steady state accuracy. Clearly, this method is able to capture the frequency evolution between segments where the AR(1) estimate failed. In this case, a step size $\mu = 0.05$ was chosen. There is a trade-off between time and frequency resolution in choosing a step size. A higher step size leads to a higher time resolution while a lower step size leads to a higher frequency resolution before not being able to converge fast enough.

3.3 A Real Time Spectrum Analyser Using Least Mean Square

Given the signal $y(n)$ the estimate using a linear combination of N harmonically related sinusoids is given by:

$$\hat{\boldsymbol{y}} = \boldsymbol{F}\boldsymbol{w} \tag{64}$$

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k) e^{j2\pi kn/N} \tag{65}$$

The vector $\boldsymbol{w}$ can be found via the minimisation of the square error:

$$\begin{aligned} ||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2 &= ||\boldsymbol{y} - \boldsymbol{F}\boldsymbol{w}||^2 \\ &= (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{w})^H (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{w}) \\ &= \boldsymbol{y}^H\boldsymbol{y} - \boldsymbol{w}^H\boldsymbol{F}^H\boldsymbol{y} - \boldsymbol{y}^H\boldsymbol{F}\boldsymbol{w} + \boldsymbol{w}^H\boldsymbol{F}^H\boldsymbol{F}\boldsymbol{w} \end{aligned} \tag{66}$$

$$\begin{aligned} \frac{d[||\boldsymbol{y} - \hat{\boldsymbol{y}}||^2]}{d\boldsymbol{w}} &= 0 - \boldsymbol{F}^H\boldsymbol{y} - \boldsymbol{F}^H\boldsymbol{y} + 2\boldsymbol{F}^H\boldsymbol{F}\boldsymbol{w} \\ &= 0 \end{aligned} \tag{67}$$

Assuming $\boldsymbol{F}^H\boldsymbol{F}$ is invertible, the LS solution is given by:

$$\boldsymbol{w} = (\boldsymbol{F}^H\boldsymbol{F})^{-1}\boldsymbol{F}^H\boldsymbol{y} \tag{68}$$

Note that $\boldsymbol{F}$ is the unitary inverse DFT (IDFT) matrix:

$$\boldsymbol{F} = \begin{pmatrix} 1 & 1 & ... & 1 \\ 1 & e^{j2\frac{\pi}{N}(1)(1)} & ... & e^{j2\frac{\pi}{N}(1)(N-1)} \\ : & : & ... & : \\ 1 & e^{j2\frac{\pi}{N}(N-1)(1)} & ... & e^{j2\frac{\pi}{N}(N-1)(N-1)} \end{pmatrix} \tag{69}$$

$$(\boldsymbol{F}^H\boldsymbol{F})^{-1} = \boldsymbol{I} \tag{70}$$

Therefore, the LS solution (68) is reduced to:

$$\boldsymbol{w} = \boldsymbol{F}^H\boldsymbol{y} \tag{71}$$

This is the IDFT formula; so the optimum weight vector is the vector of Fourier coefficients and the IDFT on this vector gives an optimal linear estimation of the original signal.

Equation (71) shows that the Fourier coefficients, $\boldsymbol{w}$, are a linear combination of the columns of the Fourier transformation matrix $\boldsymbol{F}$, which are orthonormal. Therefore, the DFT can be seen as a projection of a vector in the time domain to the $\boldsymbol{F}$ matrix subspace, representing N harmonically related sinusoids.

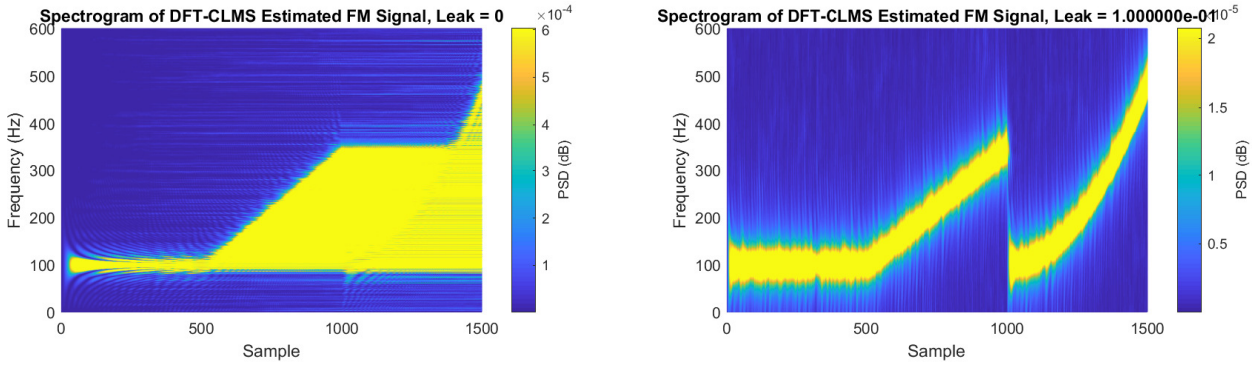


Figure 36: DFT-CLMS frequency estimation of a non-stationary signal

Figure 36 shows the DFT-CLMS frequency estimation of the same modulated FM signal in Section 3.2. In segment 1 where the frequency is constant at 100Hz, the algorithm adapts very well to the signal, obtaining a better frequency resolution when compared to the standard CLMS algorithm. However, it is clear to see that in the other segments, the algorithm appears to 'remember' its previous values and carries them through the rest of the time spectrum. This is because the weights (DFT coefficients) are calculated in an adaptive manner rather than in a block based way, and the new frequency components dominate the error variance in the gradient descent. Therefore, there is no effective back propagation. One solution is to add a leakage factor. This allows the algorithm to 'forget' previous timestep's weights. In this case, a leakage of $\gamma = 0.1$ provides a good balance between slow error backpropagation and large estimation error. However, when compared to the standard CLMS filter spectra estimation, the DFT-CLMS filter spectra estimation with leakage results in a slightly worse frequency resolution, with an overall more noisy spectrum.

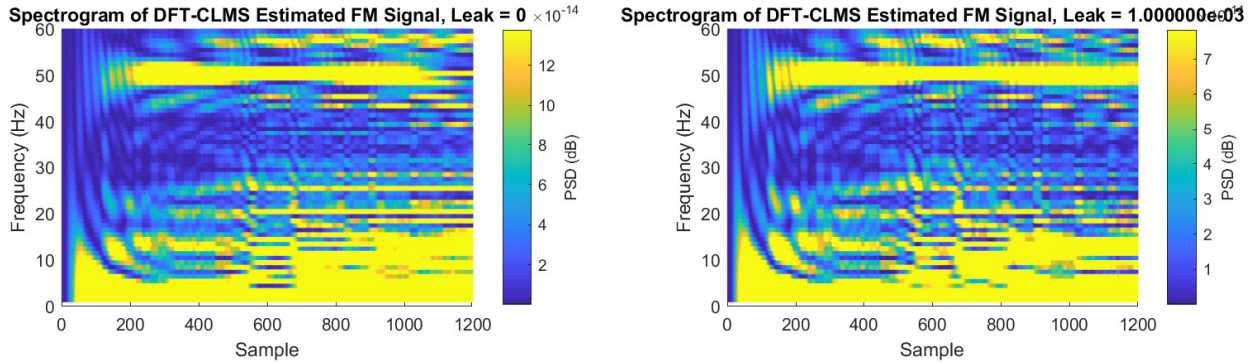


Figure 37: DFT-CLMS frequency estimation of EEG data

Figure 37 shows the DFT-CLMS frequency estimation for real EEG data. In this case, the leakage does not improve performance since the signal is stationary. The alpha-rhythm at 8-10Hz can easily be identified as can the power-line interference at 50Hz. From Section 1.2, the SSVEP signal was identified at 13Hz. This can also be identified in the DFT-CLMS spectrum but the frequency resolution is poor so it can appear to merge with the alpha-rhythm, particularly from sample 700 onwards.

4 From LMS to Deep Learning

Previously, it has been observed that the LMS algorithm is a very powerful tool for predicting non-stationary time-series in an online fashion.

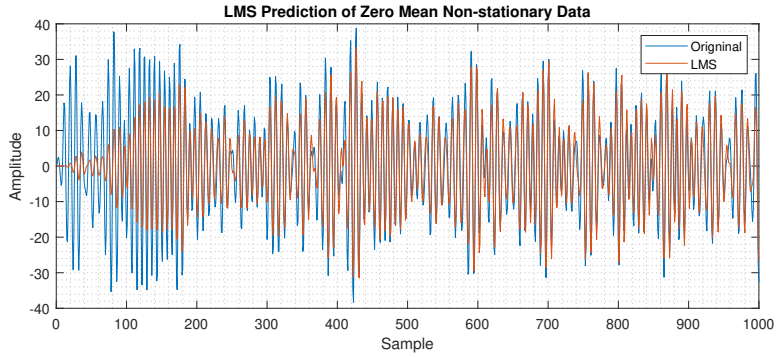


Figure 38: LMS prediction of zero mean non-stationary time-series

Figure 38 shows the LMS prediction of a zero mean non-stationary time-series with an underlying model assumption AR(4) and step $\mu = 0.00001$. The results show the output of the filter strongly following the trends of the real data after a transient of 200 samples despite mostly having a smaller amplitude. However, there is a large average error, $MSE = 40.1$ and prediction gain, $R_p = 5.196dB$.

One area for improvement is the lack of non-linearity included in the model. Since the LMS is essentially a liner regression model, it is unable to learn more complex non-linear patterns, typically a feature of most data sets. The classic example is in learning the XOR function where it is impossible to linearly separate the feature space. To improve on this, an activation function can be added to the output, which is specifically a non-linear function. The most popular in deep learning techniques is reLU which is much less computationally intensive than other functions since it is not strictly non-linear, instead being piece-wise linear, and helping to deal with vanishing and exploading gradients since its derivative is 0 or 1.

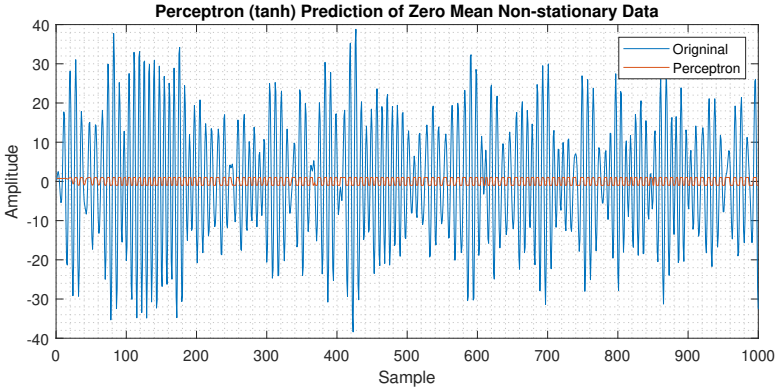
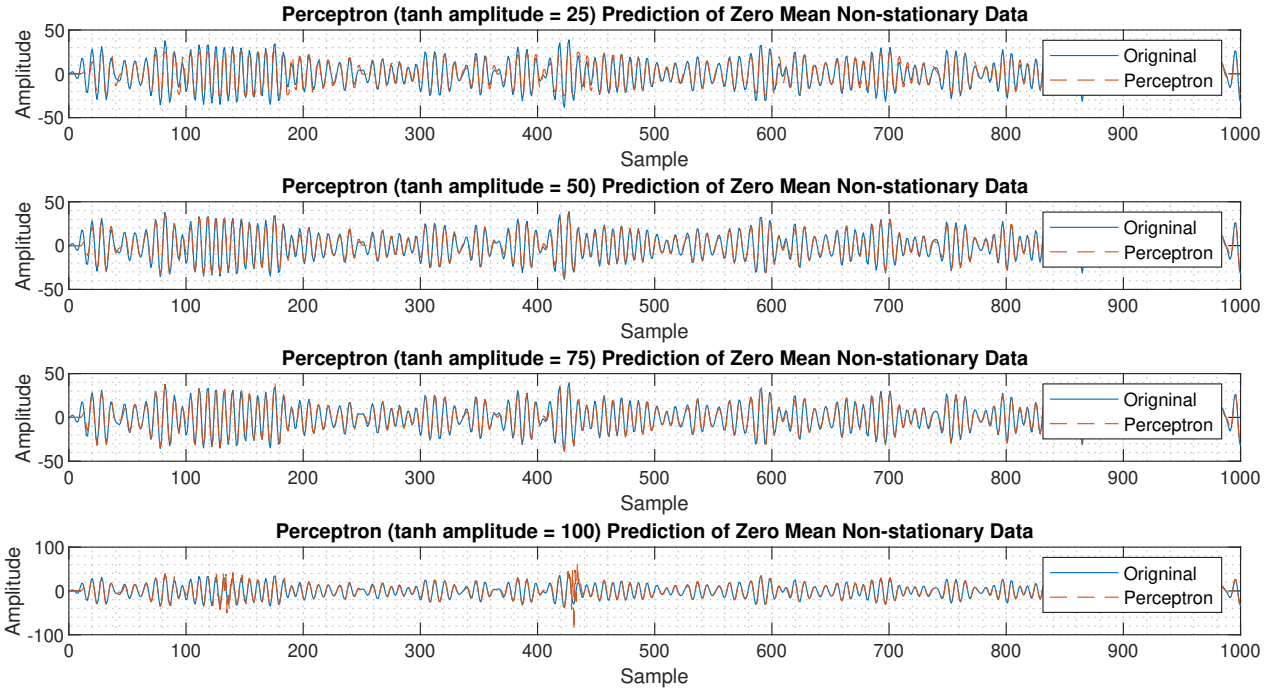


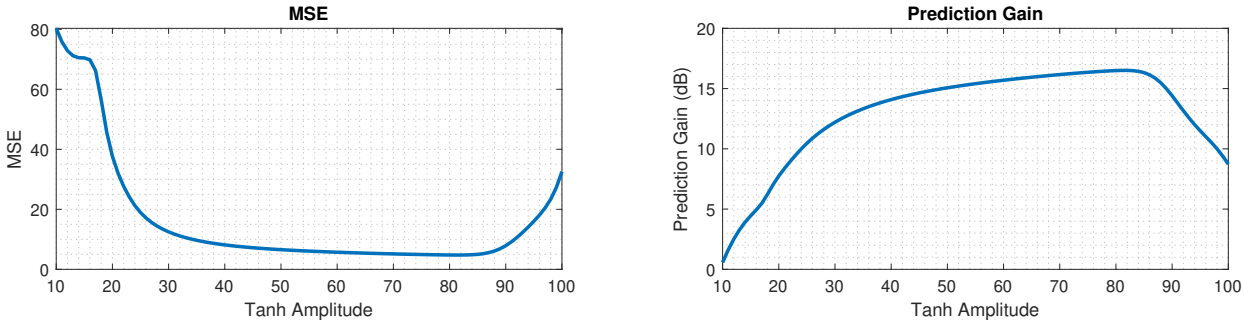
Figure 39: Perceptron prediction of zero mean non-stationary time-series using tanh activation function

Figure 39 shows the dynamic perceptron estimation of the zero mean non stationary time series. Whilst the frequency variation is successfully captured it is clear to see the amplitude of the output is capped between [-1,1]. Therefore, the average error is much higher, $MSE = 196.7$, and the prediction gain is much lower, $R_p = -23.19dB$. This is because the variance of the output is now much lower whilst the variance of the error is increased. Clearly on its own, this is not an appropriate activation function.

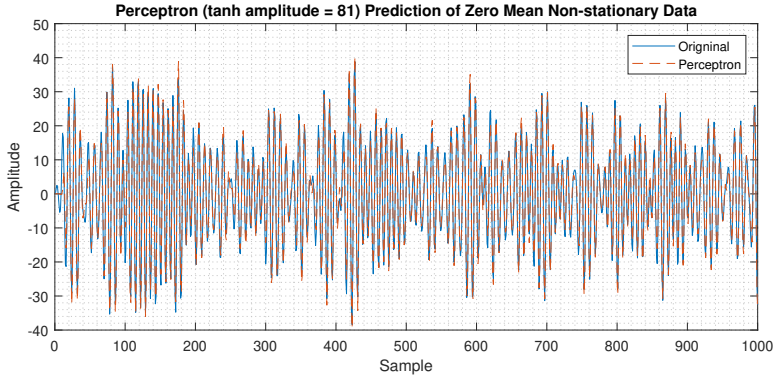
Figure 40 shows how different amplitudes of the tanh activation function affects the performance of the dynamic perceptron. At smaller values, the amplitude is capped throughout the prediction and the perceptron does not perform well. Furthermore, at very large values, glitches begin to appear where very large amplitudes are estimated which again deteriorate the performance of the model. Figure 40b shows the error curves and prediction gains for different tanh amplitudes. From these, the optimal model is found with amplitude, $a = 81$, which has average error, $MSE = 4.7699$, and prediction gain, $R_p = 16.51dB$. Finally, the optimal perceptron is modelled which appears to fit well to the original data. Clearly, this also performs better than the standard LMS; having an 88.1% smaller MSE and 218% higher prediction gain.



(a) Perceptron estimates with different tanh amplitudes

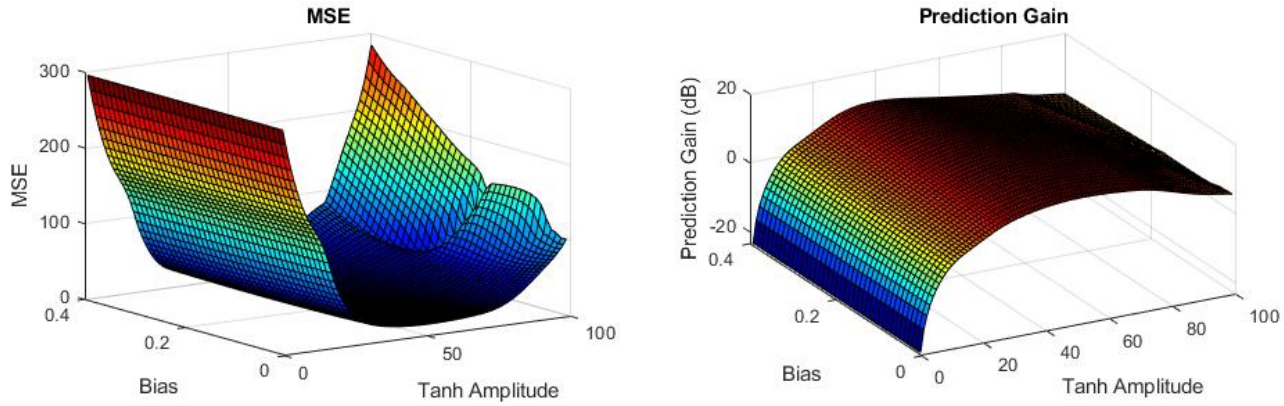


(b) MSE and prediction gain curves

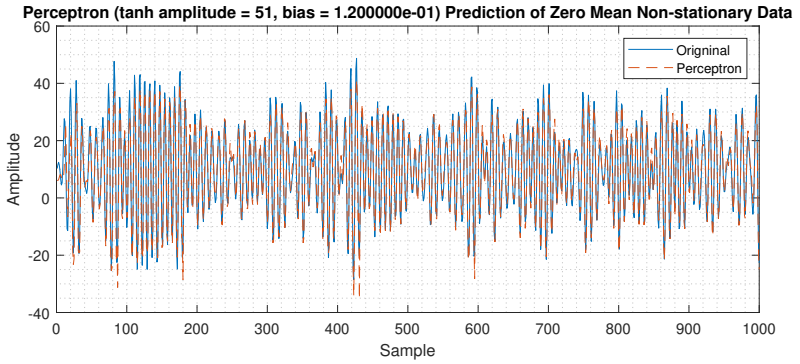


(c) Optimal tanh amplitude perceptron estimation

Figure 40



(a) MSE and prediction gain curves

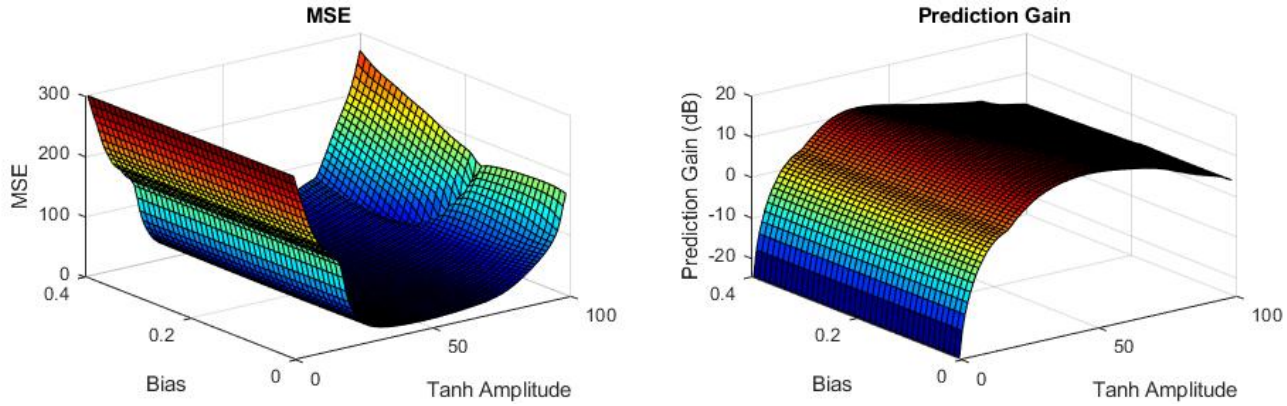


(b) Optimal tanh amplitude and bias perceptron estimation

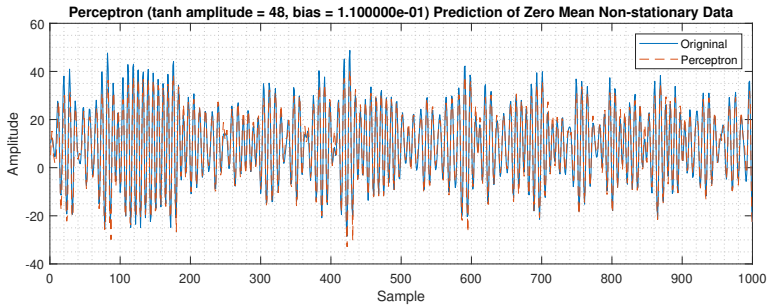
Figure 41

Figure 41 shows the performance of a dynamic perceptron with bias to predict the non-stationary time series with non-zero mean. The error surfaces in Figure 41a show an optimal result at tanh amplitude $a = 51$ and bias $b = 0.12$. Interestingly, the optimal tanh activation function amplitude is much lower than in the case of the non zero mean estimation without bias inclusion and the error and prediction gain are more correlated with the activation function amplitude than the bias. At this point, there is an error, $MSE = 10.55$, and prediction gain $R_p = 13.14dB$. These errors are worse when compared to the estimation of the zero mean stationary time series without bias inclusion. This is because the extra parameter adds complexity in trying to estimate the mean, resulting in another error contribution. However, the convergence speed of the bias included perceptron is faster than without, converging within 10 samples as opposed to 15. This is because the non-zero mean provides a positive contribution to the update of the weight in the beginning.

Figure 42 shows the dynamic perceptron estimate of the non-stationary non-zero mean time series with bias and weight initialization. The optimal tanh amplitude is now, $a = 48$ and the optimal bias is, $b = 0.11$, giving an error, $MSE = 9.54$ and prediction gain $R_p = 13.63dB$. This is a 9.57% improvement in the MSE and a 3.73% improvement in the prediction gain. The improvement here is mostly concentrated within the first 20 samples during convergence. Now, the algorithm appears to converge almost immediately, reaching steady state in one or two steps. This is clearly demonstrated in Figure 43 which shows the square error of the algorithm with weight initialization and with no weight initialization over the first 20 samples. This can be improved further by increasing the batch size, although there is a trade-off with computational complexity.



(a) MSE and prediction gain curves



(b) Optimal tanh amplitude and bias perceptron estimation with initial weights set

Figure 42

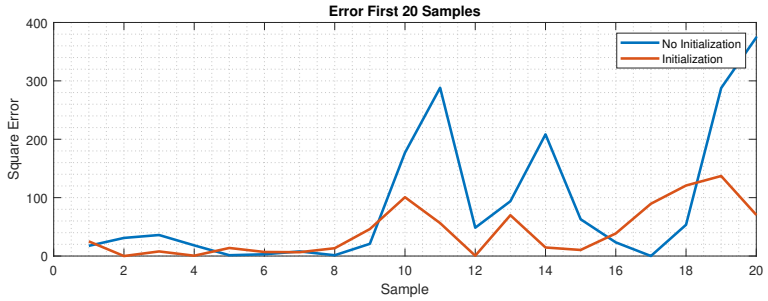


Figure 43: Error with weight initialization and no weight initialization

Perceptrons can be joined together to form layers of a deep network. The goal of such a network is to learn some function mapping the input $\boldsymbol{x}$ to the output $\boldsymbol{y}$ as $\boldsymbol{y} = f(\boldsymbol{x})$. Below is some notation associated with a deep network:

a_k^L = Activation of the k_{th} neuron in layer L
w_{jk}^L = Weight connecting the k_{th} neuron in layer L-1 to the j_{th} neuron in layer L
b_j^L = Bias of the j_{th} neuron in layer L
C_L = Total error of neurons in the last layer
$\sigma(z)$ = Activation Function

Expanding on this, let the cost function be the MSE, then looking at just the final two layers of the network, with J neurons in the last layer, L, and K neurons in the layer before, $L-1$:

$$C_L = \sum_{j=1}^{J} (a_j^L - y_j)^2 \tag{72}$$

$$a_j^L = \sigma(z_j^L) \tag{73}$$

$$z_j^L = \sum_{k=1}^{K} (w_{jk}^L a_k^{L-1}) + b_j^L \tag{74}$$

In order for the network to learn, the cost function should be minimized with respect to the network parameters (weights and biases). The chain rule can be of help here:

$$\frac{dC_L}{dw_{jk}^L} = \frac{dz_j^L}{dw_{jk}^L} \frac{da_j^L}{dz_j^L} \frac{dC_L}{da_j^L} \tag{75}$$

$$\frac{dC_L}{da_j^L} = 2 \sum_{j=1}^{J} (a_j^L - y_j) \tag{76}$$

$$\frac{da_j^L}{dz_j^L} = \sigma'(z_j^L) \tag{77}$$

$$\frac{dz_j^L}{dw_{jk}^L} = \sum_{k=1}^{K} a_k^{L-1} \tag{78}$$

$$\frac{dC_L}{dw_{jk}^L} = 2 \sum_{j=1}^{J} (a_j^L - y_j) * \sigma'(z_j^L) * \sum_{k=1}^{K} a_k^{L-1} \tag{79}$$

Equally for the bias:

$$\frac{dC_L}{db_j^L} = \frac{dz_j^L}{db_j^L} \frac{da_j^L}{dz_j^L} \frac{dC_L}{da_j^L} \tag{80}$$

$$\frac{dz_j^L}{db_j^L} = 1 \tag{81}$$

$$\frac{dC_L}{db_j^L} = 2 \sum_{j=1}^{J} (a_j^L - y_j) * \sigma'(z_j^L) \tag{82}$$

It can be seen that these derivatives can be obtained as information passes through the network in training. Knowing the derivative of the cost with respect to the network parameters of the final layer means these can be updated via the gradient descent algorithm which is a globally convergent algorithm (optimal choice over Newton based algorithms) since the negative gradient is always a descent direction satisfying the condition of angle:

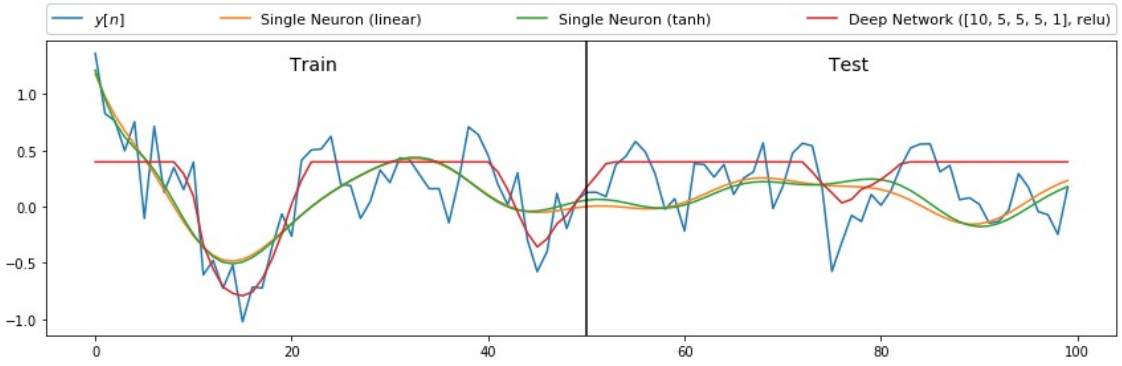
$$w_{jk}^L(t+1) = w_{jk}^L(t) - \mu \frac{dC_L}{dw_{jk}^L} \tag{83}$$

$$b_j^L(t+1) = b_j^L(t) - \mu \frac{dC_L}{db_j^L} \tag{84}$$

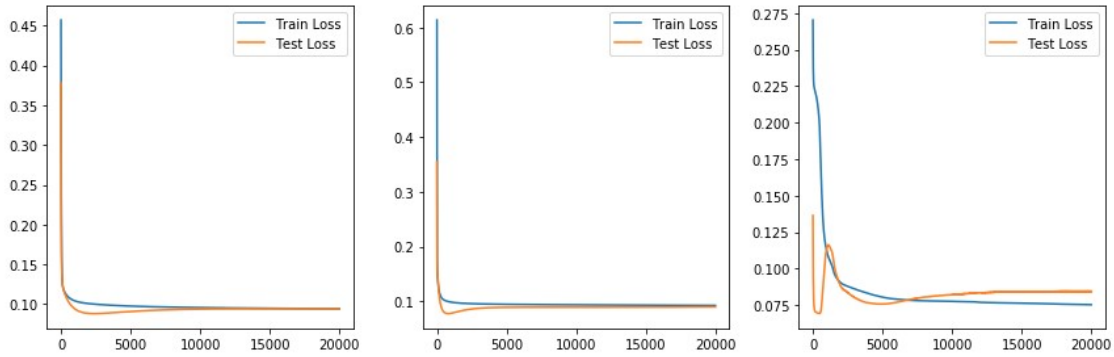
Where μ is the learning rate which should be chosen to be small and positive to ensure convergence (can be chosen via line search parameter estimation such as Armijo algorithm). The idea of backpropagating comes in when the derivative of the cost function with respect to a previous neuron activation can also be calculated using the chain rule in the same way as before, for example, with respect to the activation of a neuron in the previous layer, $L-1$:

$$\frac{dC_L}{da_k^{L-1}} = 2 \sum_{j=1}^{J} (a_j^L - y_j) * \sigma'(z_j^L) * \sum_{k=1}^{K} w_{jk}^L \tag{85}$$

Whilst this cannot be used to directly impact the activation, it is useful to keep track of as it allows the chain rule algorithm to be iterated backwards throughout the entire network, updating weights and biases on the way using the gradient descent algorithm.



(a) Predictions of different models on y[n]

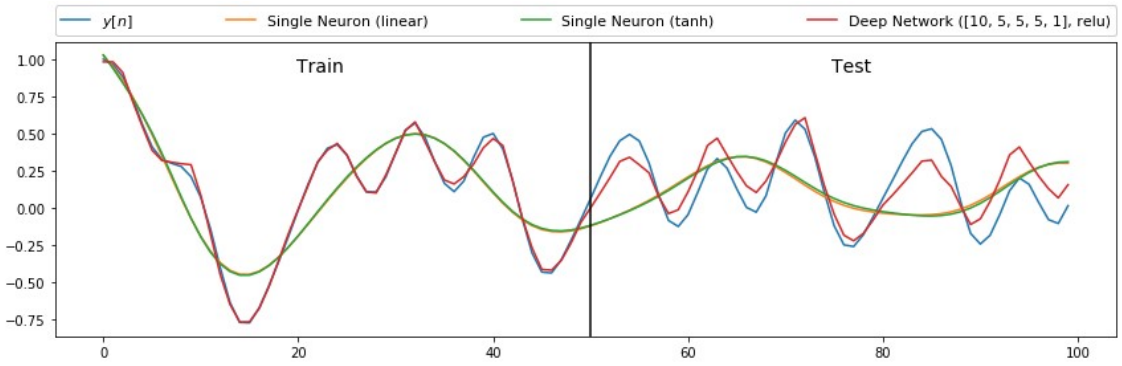


(b) Performance of the linear neuron, tanh neuron and deep network respectively

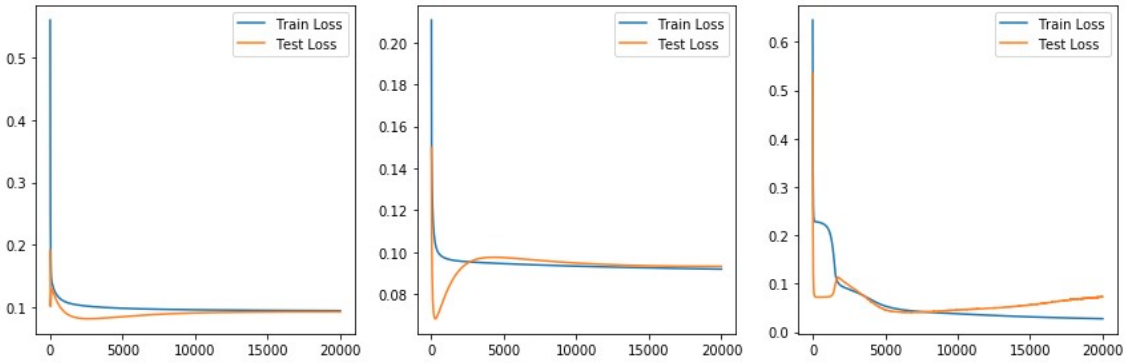
Figure 44

Figure 44 shows the performance of various predictors on the highly non-linear signal $y[n]$. The deep network used to train on this time series has layers, epochs and learning rate given by $[L, \epsilon, \mu] = [4, 20000, 0.01]$ and noise variance is $\sigma_n^2 = 0.05$. Surprisingly, both the linear and tanh activated neurons have similar performance, with a steady state loss at 0.1 despite the large degree of non-linearity in $y[n]$. The main difference between the two single neurons is that the tanh neuron converges faster within 5000 epochs whilst the linear neuron converges within 10000 epochs. Both the training and test losses are the same in the steady state. The deep network, on the other hand, has a smaller steady state loss at 0.075 for the training set and 0.08 to 0.085 for the test set. However, it takes longer to converge than both the single neurons, within 15000 epochs. However, this is to be expected with its greater complexity. The prediction graph indicates that overfitting is not an issue here, with good generalization in the training phase, ignoring the individual random variations of the signal.

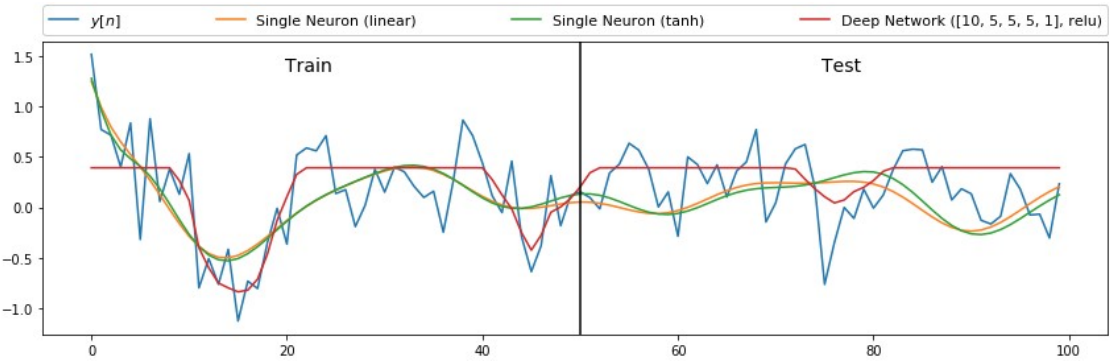
Figure 45 compares the performance of various predictors on the highly non-linear signal $y[n]$ with varying noise power. As expected, across all three models, the performance for the signal with zero noise power (noise variance) is better than for the signal with high noise power.
In the high SNR case of Figure 45a and Figure 45b, the deep network performs best with the test loss remaining below 0.1 after around 2500 samples and reaching a minimum of around 0.05. Again the training loss is even lower, reaching around 0.01. The tanh activated perceptron performs slightly better than the linear perceptron, with both training and testing losses converging to just below 0.1 whereas the linear perceptron converges closer to a loss of 0.1. Whilst the training loss for both algorithms appears to converge within 5000 samples, the test loss for both models converges at closer to 15000 samples. The neural network never appears to really converge to a steady state value. In fact, the diverging test and training error curves indicate the deep network is overfitting to the training data. This is further supported by how the deep network prediction curve almost entirely overlaps the signal in the training phase. In reality, early stopping should be used to stop training at around 6000 epochs where the test loss is at a minimum.
In the low SNR case of Figure 45c and Figure 45d, the deep network performs best with a steady state test loss of 0.1. The tanh activated perceptron and linear perceptrons perform very similary, with a steady state test loss of around 0.13. However, the tanh model converges much faster within around 7500 epochs compared to 17500 epochs. The deep network sits in the middle, converging within 10000 epochs. The training loss in all three cases is larger than the test loss, with the difference being largest in the deep network. Furthermore, the network appears to not overfit with parallel train and test losses as well as good generalization in the training period.
Overall, the deep network provides better performance than the perceptron models; however, this comes at the cost of computational complexity and care must be taken to avoid overfitting.



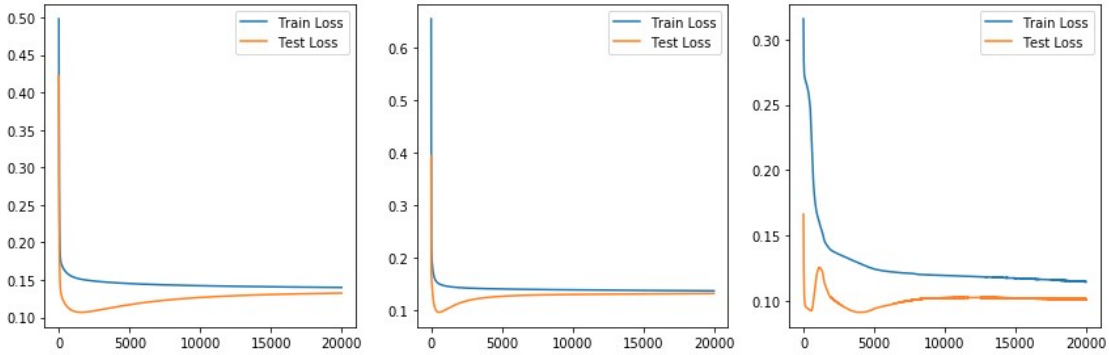
(a) Predictions of different models on y[n], $\sigma_n^2 = 0$



(b) Performance of the linear neuron, tanh neuron and deep network respectively, $\sigma_n^2 = 0$



(c) Predictions of different models on y[n], $\sigma_n^2 = 0.1$



(d) Performance of the linear neuron, tanh neuron and deep network respectively, $\sigma_n^2 = 0.1$

Figure 45

5 Tensor Decompositions for Big Data Applications

This section is completed in Jupyter Notebooks (see Part 5 file directory).