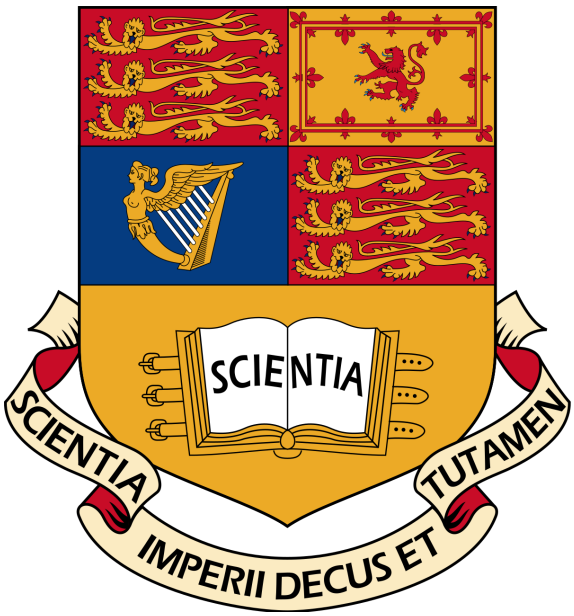
Imperial College London

Department of Electrical and Electronic Engineering

Adaptive Signal Processing and Machine Intelligence



Author
Ricardo Vera

CID
01061547

Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

Approximation in the definition of PSD

The Power Spectral Density can be defined as:

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

In order to get rid of the double summation and leave the expression only in terms of k,an observation is needed on the effect of the first summation. To get rid of the n summation we need to realise that its behaviour signifies a sliding window across the k for every n, such that k will occur in a triangular manner, with 0 occurring most (N times) whilst $N-1$ and $-N+1$ occurring only once, hence giving the relation $N-|k|$. Hence the result of Equation 1 can be expressed as:

$$\begin{aligned}
&= \lim_{N\to\infty} \frac{1}{N}\sum_{k=-N+1}^{N-1} (N-|k|)r(k)e^{-j\omega(k)} \\
&= \lim_{N\to\infty} \frac{1}{N}\left(\sum_{k=-N+1}^{N-1} Nr(k)e^{-j\omega(k)} - \sum_{k=-N+1}^{N-1} |k|r(k)e^{-j\omega(k)}\right)
\end{aligned} \tag{2}$$

Which under a mild assumption that the covariance sequence r(k) decays rapidly, equation 3, proves the equality in Equation 4.

$$\lim_{N\to\infty} \frac{1}{N}\sum_{k=-N+1}^{N-1} |k|r(k) = 0 \tag{3}$$

$$P(\omega) = \lim_{N\to\infty} \mathbb{E}\left\{\frac{1}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-jn\omega}\right|^2\right\} = \sum_{k=-\infty}^{\infty} r(k)e^{-j\omega k} \tag{4}$$

1.2 Periodogram-based Methods Applied to Real–World Data

Applying the periodogram to the sunspot data with different preprocessing shows different points of interest in the resulting spectrum estimation, Figure 1. Removing the `mean` from the signal removes the DC component whilst performing the `detrend` operation removes low frequency components driving the signal. Little effect do both operations have on the rest of frequencies,

hence the plot for both can be seen to be identical except for the lower frequencies for which the operations have considerable effect.

By performing the log of the data and then removing the mean a very different plot is obtained. The effect of the log function is to compress the data, magnitudes over 1 are attenuated resulting in a much smoother spectrum where the key peaks are easily distinguished, with the final mean subtraction removing the DC component. Note to avoid log of zero a small constant (MATLAB eps) is added to the data.

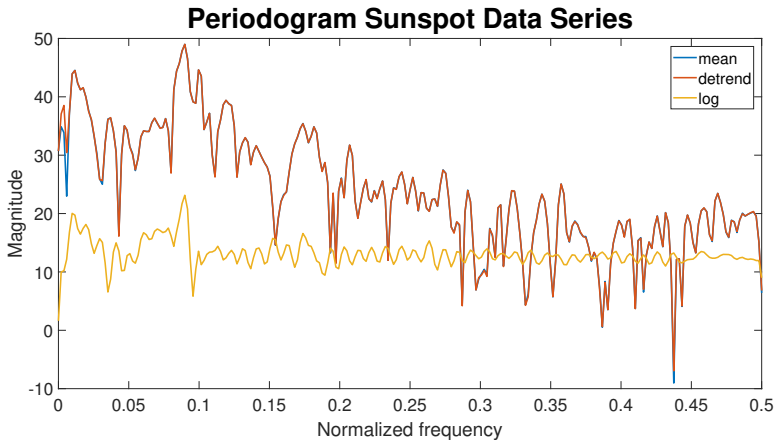
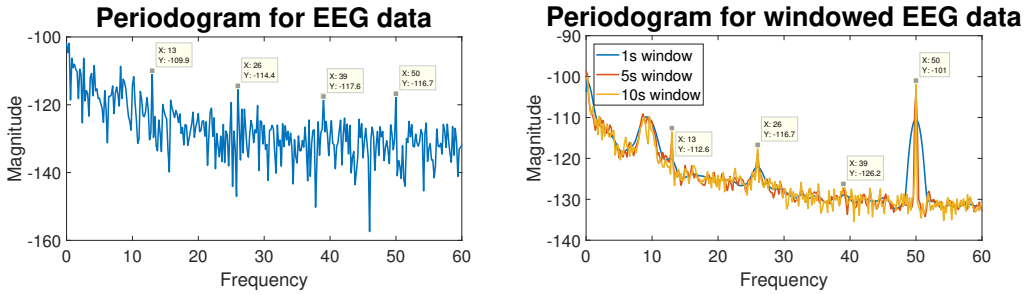


Figure 1: Periodogram of sunspot data for different preprocessing

The basis for brain computer interface (BCI)

Applying the periodogram on EEG data we can aim to detect the frequency of visual stimulus with a constant frequency, which produces a response in the EEG, the steady state visual evoked potential (SSVEP), of the same frequency. This frequency is clearly identifiable from Figure 2 at 13Hz, with the averaged procedure showing clearer peaks for identification. As told clear peaks show at $8-10$HZ due to the subject's tiredness and at 50Hz representing the power line interference.



(a) Standard Periodogram (b) Averaged Periodogram for varying windows

Figure 2: Standard and Averaged Periodograms of EEG

Comparing the standard periodogram against the averaged procedure is a trade-off between resolution and variance. The longer the window the lower variance hence reducing noise around the frequencies of interest, however resolution is reduced thus leading to spectrum estimation errors. As can be seen from Figure 2a the peaks are far less recognisable that in Figure 2b with a window length of 10s giving an easy identification of the frequency harmonics. As such an averaged periodogram of window length 10s is better for SSVEP identification than the standard periodogram. However, Figure 2b shows how resolution is also to be considered for the task, for window length 1s the 3^{rd} harmonic is not recognisable, as the reduced variance isn't enough to cope with the loss of resolution, leading to errors.

1.3 Correlation Estimation

Figure 3 shows the *biased* and *unbiased* autocorrelation function (ACF) estimates as well as the correlograms: White Gaussian Noise (WGN), a noisy sinusoid and filtered WGN.

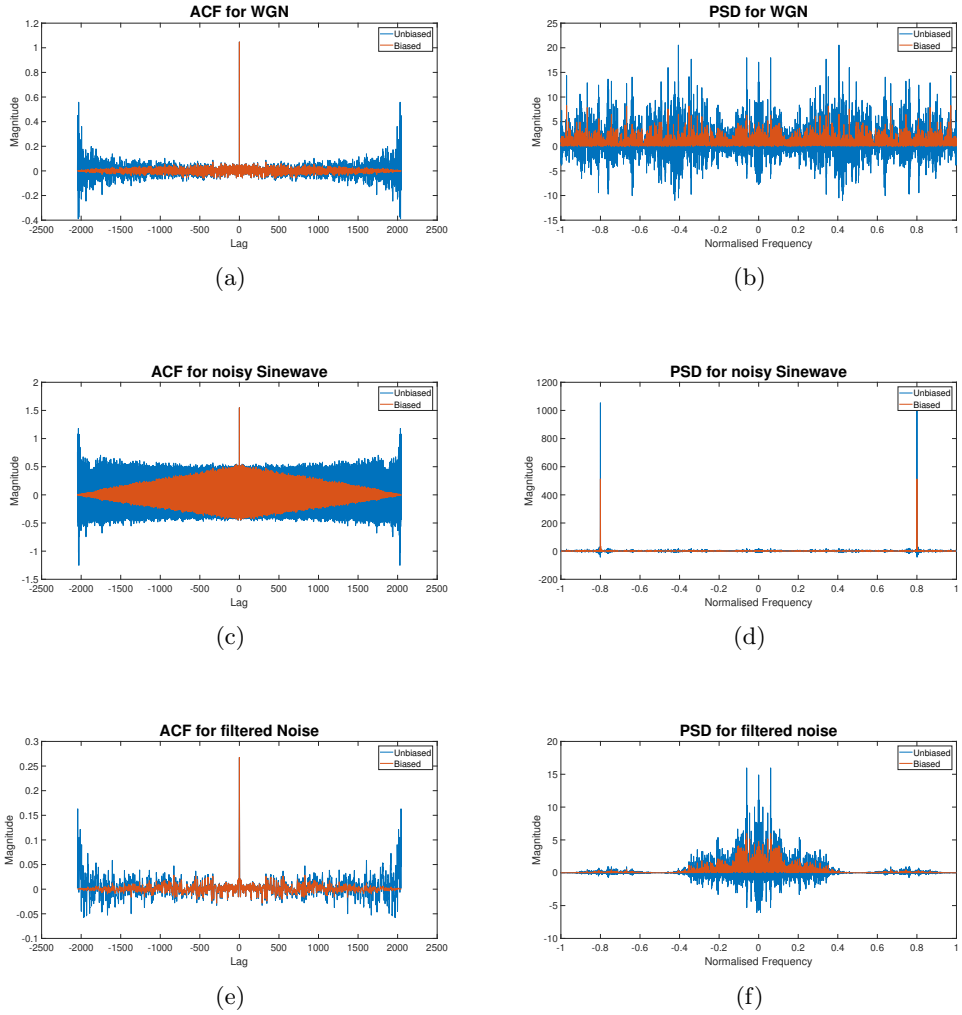


Figure 3: ACF and PSD for WGN, noisy Sinewave and filtered WGN

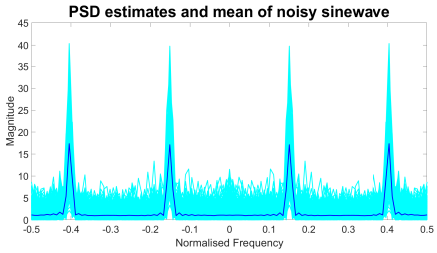
The ACF for noise should be 0 everywhere except for $lag = 0$, this is true for the biased version, Figures 3a and 3e, however the unbiased method increases with lag. This is due to the number of samples being considered reducing and making the calculation unreliable. This also happens in Figure 3c, whilst the biased version decreases linearly with lag. The biased version should have reciprocal behaviour, however for the small amount of samples considered this seems linear as shown in Figure 3c. The different window forms result in the different behaviour, with the biased version attenuating values far from the window centre, and thus fading to zero for large lag values. The issues posed by the inaccuracies of the unbiased method produce negative PSD values which is clearly not possible.

The difference in windows has a direct effect on the PSD. The *rectangular* window in the unbiased procedure produces a *sinc* function in the frequency domain, whilst the biased method produces a *triangular-like* function which results in a $sinc^2$ function in frequency. From this we can observe why the unbiased version has negative values, introduces by the *sinc* function, which the biased version doesn't have due to the positive nature of the $sinc^2$.

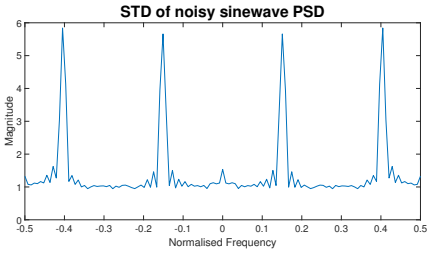
Figure 4 shows the PSD of a noisy sinusoid, equation 5, for various realisations, with the mean PSD shown in dark blue and Figure 4b showing the standard deviation at every frequency. As can be seen the maximum deviation happens at then main points of interest around the frequencies

of the sinusoid. This is confirmed by comparing the difference in magnitude of the mean and all observations, it is clear the deviation is higher at the sinusoid's frequencies with noise having less effect at other spectrum areas.

$$x(n) = sin(2\pi 0.4n) + sin(2\pi 0.15n) + w(n) \qquad w(n) \sim \mathcal{N}(0,1) \tag{5}$$



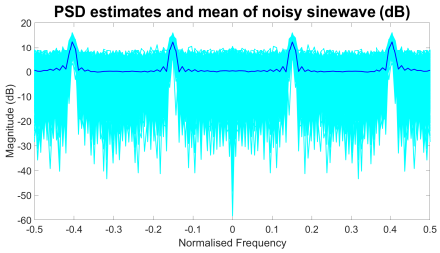
(a) PSD observations and Mean



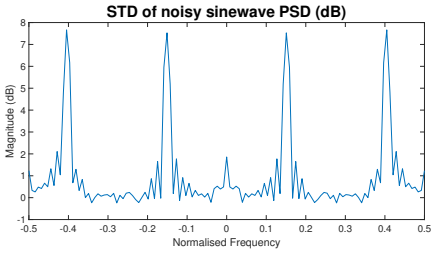
(b) PSD standard deviation

Figure 4: PSD mean and std for various realisations

If we plot the above figures in dB, we obtain Figure 5. From these plots we can see how the periodogram is somehow more clear with the main lobes being more recognisable. The standard deviation shape doesn't change much as expected, however the uncertainty for noise frequencies (ie, not the ones of the signal) shows greater variations outlying the differences in noise at different frequencies.



(a) PSD observations and Mean



(b) PSD standard deviation

Figure 5: PSD mean and std for various realisations

The periodogram resolution is proportional to $\frac{1}{N}$, hence the larger the number of points the more precise the spectrum estimation by the periodogram. Figure 6 shows the different spectrum estimates as the number of points is varied. The signal is composed of two frequencies with 0.02Hz difference (0.3 and 0.32HZ), considering the resolution's proportionality to $\frac{1}{N}$, 50 samples would be needed. However from Figure 6 we can see how for $N = 35$ the two frequencies already start to show, implying the proportionality is slightly below $\frac{1}{N}$ and more something like $\frac{0.8}{N}$, which would require 40 samples. From Figure 6 it is clear for $N = 50$ the two are very clearly distinguishable and lower N would suffice.

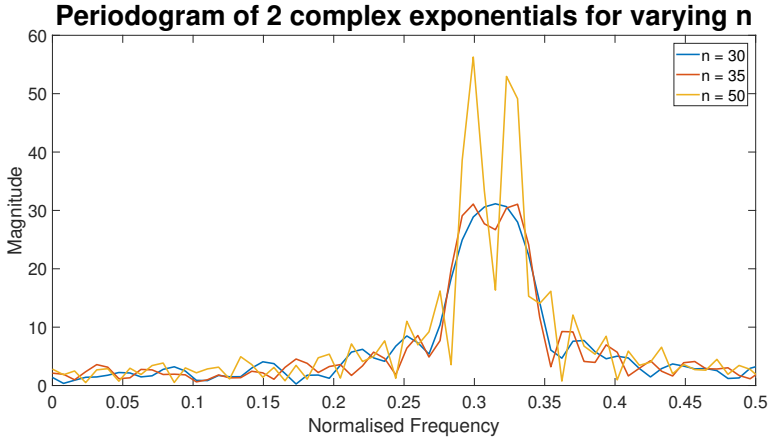


Figure 6: Periodogram of sinusoid for varying number of points N

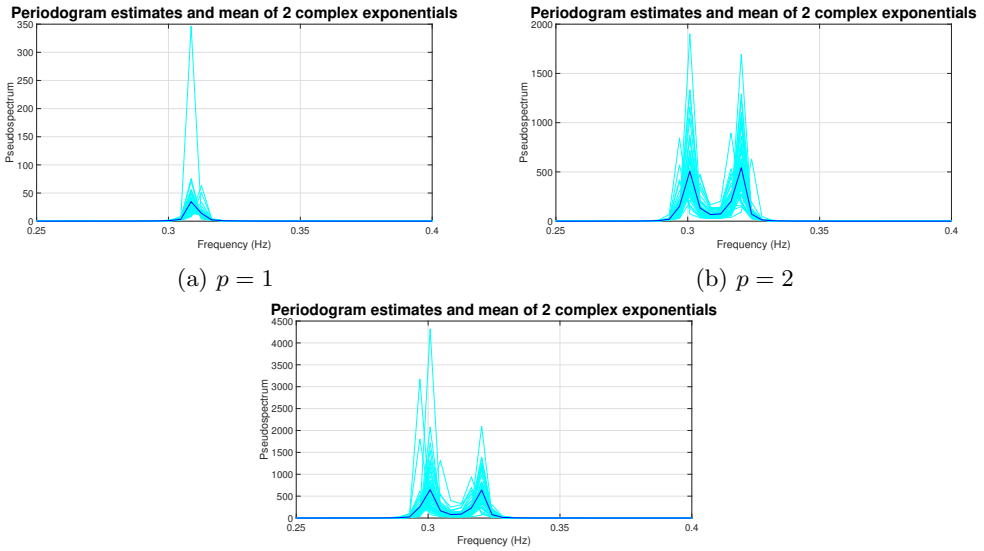
Finally, in order to perform frequency estimation of a signal we can use the MUltiple SIgnal Classification (MUSIC) algorithm. The MUSIC algorithm is a Signal-Subspace type technique which assumes the decomposition of the signal into p complex exponentials in the presence of noise. As such eigenvalue decomposition can be performed to the signal's autocorrelation matrix to obtain the signal and noise subspaces from the magnitudes of the eigenvalues (largest eigenvalues being those of the signal). With the eigenvalues providing a means for sorting, $\mathbb{R}_{xx}$ can be decomposed into $\mathbb{R}_S$ and $\mathbb{R}_n$ for the signal and noise representation respectively. Thus, if we define $\mathbf{v}_i$ as the noise eigenvectors and $\mathbf{e} = [1, e^{j\omega}, e^{j2\omega}, \ldots, e^{j(M-1)\omega}]^T$ (where M is the number of samples), the MUSIC algorithm frequency estimation is defined by:

$$\hat{P}_{MU}(e^{j\omega}) = \frac{1}{\sum_{i=p+1}^{M} |\mathbf{e}^H \mathbf{v}_i|^2} \tag{6}$$

The result of equation 6 is p peaks signifying the p complex exponentials composing the signal. The complex exponentials of the signal will have null magnitude in the projected noise subspace, thus the reciprocal form of the equation will project these points to peaks outlining their orthogonality to the noise subspace.

In order to perform the MUSIC algorithm we first perform the autocorrelation operation with a lag of 14 to avoid the issues arising for bigger values of lag, as we are using $N = 30$. Then the MUSIC algorithm is used, defining p as the number of complex exponentials, to output the pseudospectrum and frequency range.

```
[X,R] = corrmtx(x,14,'mod');
[S,F] = pmusic(R,2,[ ],1,'corr');
```



(a) $p = 1$ (b) $p = 2$

(c) $p = 3$

Figure 7: Spectrum estimation using MUSIC and varying p

Figure 7 shows the performance of the MUSIC algorithm for varying values of p. The plots show the mean estimated spectrum (blue) as well as the 50 observations (cyan) from which it is computed. From the figures it can be seen how the MUSIC algorithm struggles if the correct p is not selected, producing non-ideal spectra. The method does however perform very successfully for the correct $p = 2$ estimating the frequencies for only 30 samples, unlike previously seen in Figure 6 where with $N = 30$ it was impossible to distinguish both peaks. As such the main advantage of the MUSIC algorithm is its ability to correctly identify the complex exponentials even with few samples, unlike the periodogram.

This method however has various disadvantages. The spectrum's variance is considerable, as can be seen comparing different realisations, thus providing uncertainty in the spectrum estimation. Additionally, this method doesn't provide accurate information on the peaks' magnitude, the exponentials have equal amplitude, whilst in the estimated spectra the amplitudes differ. The mean spectrum does somehow compensate this, but within realisations the spectrum offers and incorrect spectrum amplitude, thus making unreliable for this matter. Finally, the method needs the prior knowledge of the parameter p for correct estimation. As can be seen in Figures 7a and 7c there is large deviation when p is incorrect, p could be estimated via AIC or MDL methods (as is in areas such as Communication Systems) but this would require an additional step for the algorithm, which would introduce complexity considerations.

To conclude the MUSIC algorithm provides a very good alternative to estimate the frequencies of a certain signal specially for low values of N, giving clear localisation of the desired frequencies. However, its drawbacks with respect to variance and magnitude inaccuracy limit it to a frequency localiser and not the desired spectrum estimator.

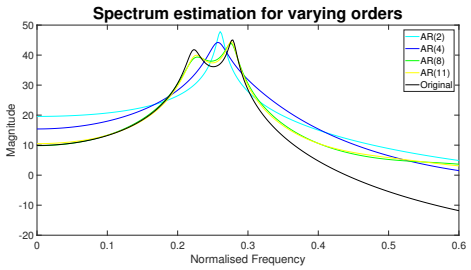
1.4 Spectrum of Autoregressive Processes

To estimate the coefficients of an Autoregressive Process we perform $\mathbf{a} = \mathbb{R}_{xx}^{-1}\mathbf{r}_{xx}$, which relies on being able to invert the autocorrelation matrix. Using the biased form of the ACF guarantees the matrix being positive definite and hence invertible, however the unbiased form doesn't guarantee this meaning the inverse may not be possible to compute.

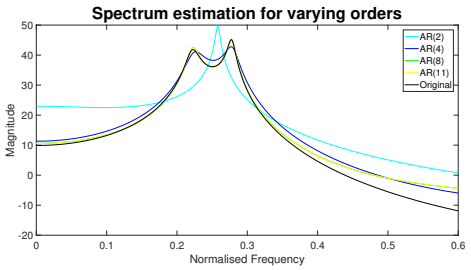
The autoregressive process defined by equation 7 is produced, using estimation techniques different order coefficients are estimated aiming to estimate correctly the process' spectrum.

$$x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n) \qquad w(n) \sim \mathcal{N}(0,1) \quad (7)$$

Figure 8a shows the spectrum estimation for varying model orders using 1000 points. As can be seen lower order models do not manage to correctly capture both peaks and give inaccurate estimates. As model order increases the spectrum estimation improves with AR(8) and AR(11) giving very good estimates just missing on magnitude. The theoretical model order is AR(5) (not shown) which however had large deviation in terms of magnitude.



(a) Number of points = 1000



(b) Number of points = 10000

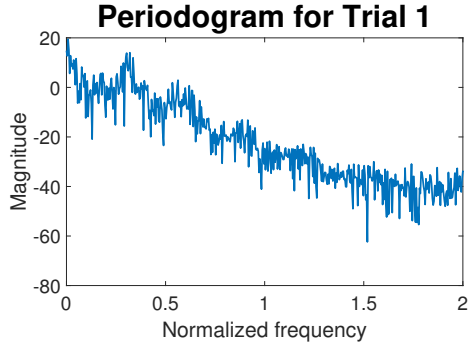
Figure 8: Spectrum estimates for varying order and number of points

Figure 8b shows the repetition of the above procedure, however using 10000 points instead of 1000. The larger amount of points available improves significantly the accuracy of the estimates. Only AR(2) does not estimate to a degree the correct spectrum and models above AR(6) performing almost perfect estimation. A clear trend is also outlined in Figure 8b, the tendency to overfit by the models. The complexity in the main area surrounding the main frequencies makes higher order models overfit, however as can be seen at frequencies above 0.5, the difference with the original is significant. The models try to get the main area right and so overfit in other areas of the spectrum, realise how the spectrum estimate of AR(11) is above that of AR(4) for $f > 0.5$.

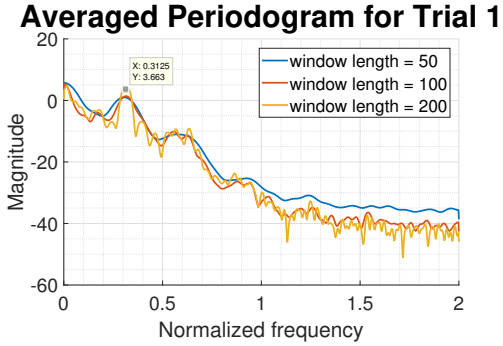
1.5 Real World Signals: Respiratory Sinus Arrhythmia from RR-Intervals

We now study Respiratory sinus arrhythmia which relates to a change in heart rate related to a constraint in breathing frequency. Using real data from measured ECG under various breathing conditions, we get our RRI data to be examined. Using the Hamming window the standard periodogram is performed as well as averaged periodograms of varied length to examine the trade-off between resolution and variance.

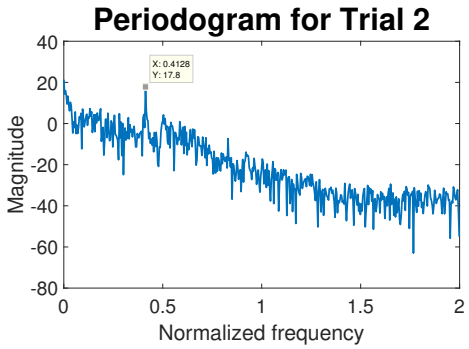
Figure 9 shows the standard periodogram and averaged periodogram using windows of length 50, 100 and 200 samples, of the 3 different RRI trials. The first trial shows a peak at ~ 0.30Hz symbolising the unconstrained breathing pattern. Trial 2 shows the peak at the expected 0.4129Hz which shows the breathing pattern of 50BPM, $BPM = 2 \times 60 \times f = 49.55$. Finally trial 3 shows the peak expected again at 0.1206Hz for the breathing pattern of 15BPM, $BPM = 2 \times 60 \times f = 14.47$.



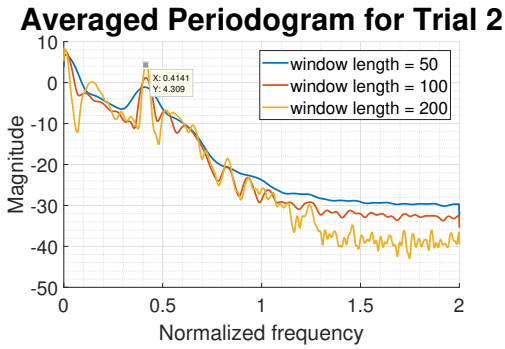
(a) Trial 1 original data



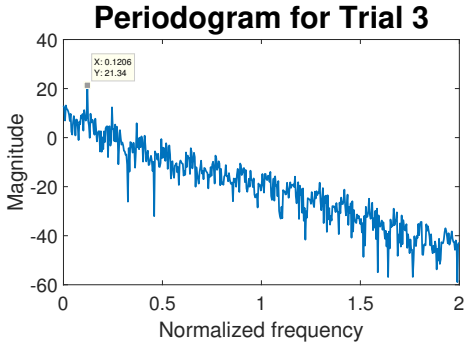
(b) Trial 1 averaged data



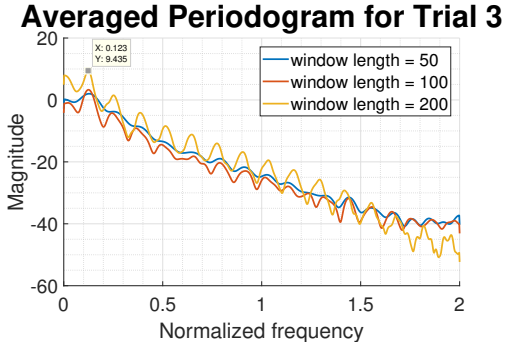
(c) Trial 2 original data



(d) Trial 2 averaged data



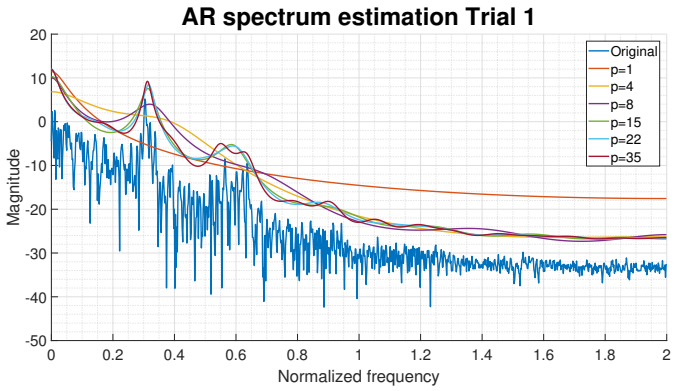
(e) Trial 3 original data



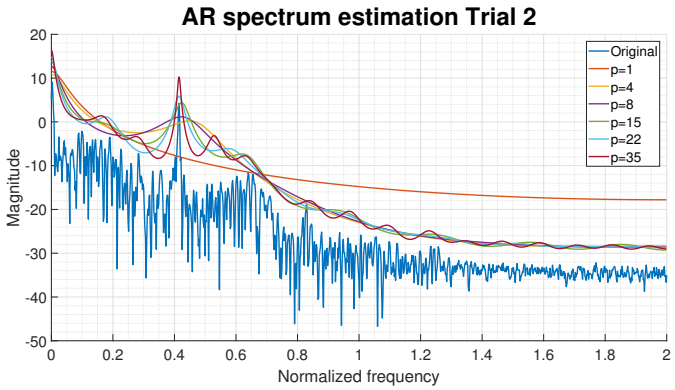
(f) Trial 3 averaged data

Figure 9: Periodograms of original and averaged RRI data

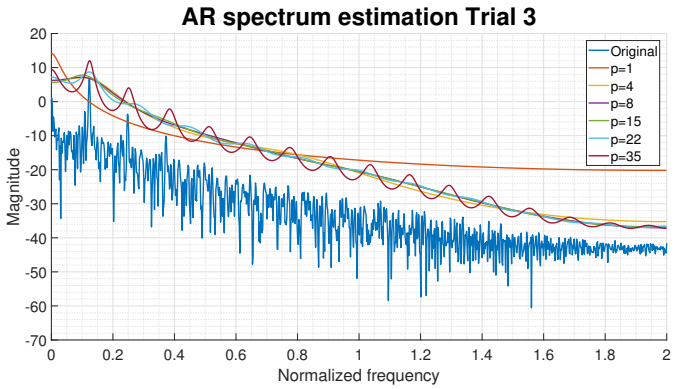
As can be seen from Figure 9 the averaged periodogram reduces variance significantly, making the plots much smoother as window size decreases. It is worth remembering the averaged periodogram is a trade-off between resolution and variance, as can be seen for the window length of 50 samples becoming extra smooth and being unable to confidently identify the harmonics of the breathing frequency. Window length of 200 samples however provides too much variance as can be seen clearly in Figure 9c where the peak is actually broken in two and hence doesn't identify the frequency accurately. In this case window length of 100 samples provides a good in-between of both, performing successfully in both areas.



(a) Trial 3 estimated spectra



(b) Trial 3 estimated spectra



(c) Trial 3 estimated spectra

Figure 10: AR estimates of ECG data for varying order model

Finally spectrum estimation is performed on the ECG data using AR models. It is clear from all plots that the estimated spectrum is very much conditioned by the need to fit the peak accurately giving a significant distance for the rest of the spectrum even for high order models.

For trial 1 model order above AR(8) seems to be sufficient, with order models above 15 trapping the harmonics, and model orders above 22 overfitting the curve at the harmonics. The spectrum estimation is quite successful at identifying the peak for not very high orders, with the main aim being identifying the main peak orders above 8 are a sufficiently successful approach.

Trial 2 poses a bit more of a challenge for spectrum estimation. Again higher order models do achieve correct estimation and again tend to overfit. In this case satisfactory order models are good enough from order 15 onwards, though as shown order 8 does show the general trend.

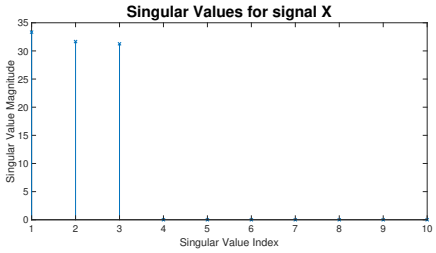
Trial 3 presents a similar scenario to trial 1 where the peak is correctly estimated by almost all models, however the rest of the spectrum exhibits inaccurate estimation. Order 35 clearly shows

the overfitting of higher order models whilst models below order 20 identify the frequency peak but fail to reproduce the actual peak.

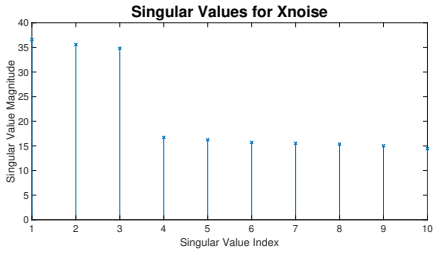
Overall it can be seen how AR models produce very smooth spectrum estimates of the ECG data proving their appropriateness for the task. Using methods such as the periodogram aim at reducing the variance in order to highlight the peaks in the spectrum, but suffer from issues regarding the correct choice of windowing. AR models on the other hand provide a smoother spectrum, easy for peak identification, but that may be slightly more inaccurate in terms of spectrum magnitude as seen. The issue with AR spectrum estimation lies more in the challenge of order choice in order to give an accurate spectrum representation whilst avoiding overfitting.

1.6 Robust Regression

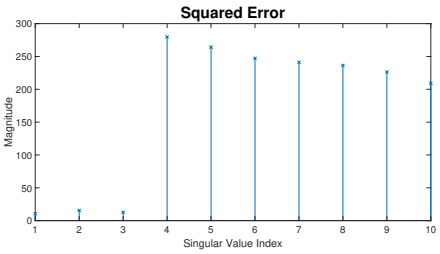
Figure 11 shows the eigenvalues for $\mathbf{X}$ and $\mathbf{X}_{noise}$. From Figure 11a it is clear $\mathbf{X}$ is rank 3, with the remaining values being zero. The eigenvalues of $\mathbf{X}_{noise}$ show a similar thing with the added magnitude from noise. The rank is still recognisable but the remaining values are no longer zero and the magnitude of the first 3 has an increased magnitude.



(a) Eigenvalue for X



(b) Eigenvalue for X_{noise}



(c) $\mathbf{X}$ and $\mathbf{X}_{noise}$ squared error

Figure 11: Eigenvalue magnitudes

Figure 11c shows how the effect of noise has much less effect on the signal's eigenvalues with the error being significantly lower. If noise magnitude was greater this difference could reduce making it harder to deduce the rank.

With the knowledge of the matrix rank we can perform Principal Component Analysis (PCA), on the noisy matrix, aiming to get rid of the noisy eigen-components and hence reconstruct the original noiseless matrix. Figure 12 shows the squared error, defined as $|\mathbf{X} - \mathbf{X}_{\mathbf{recon}}|_F^2$, of the reconstructed matrix according to which rank the matrix is assumed to be. It is clear $Rank = 3$ leads to a minimum error as it is, as shown by previous plots, the actual rank of the signal. The difference between the reconstructed matrix and $\mathbf{X}_{noise}$ decreases with rank as expected. However it is clearly seen how the error decreases sharply until rank 3, when most of the information is already present, from when increasing rank is just adding noise components.

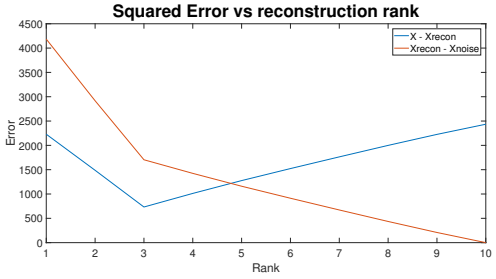


Figure 12: Squared error vs rank

The output data is obtained as $\mathbf{Y} = \mathbf{XB} + \mathbf{N}_Y$, hence the objective is to obtain $\mathbf{B}$ such that we can generate $\mathbf{Y}$ just with the error concerning to noise. This can be done in two ways: Ordinary Least Squares (OLS) and Principal Component Regression (PCR). Both solutions to matrix $\mathbf{B}$ are defined as:

$$\begin{aligned} \hat{\mathbf{B}}_{OLS} &= (\mathbf{X^T X})^{-1}\mathbf{X^T Y} \\ \hat{\mathbf{B}}_{PCR} &= \mathbf{V}_{1:r}(\mathbf{\Sigma}_{1:r})^{-1}\mathbf{U}_{1:r}^T\mathbf{Y} \end{aligned} \tag{8}$$

Since $\mathbf{X}^T\mathbf{X}$ in OLS has to be invertible $\mathbf{X}_{noise}$ is used instead of $\mathbf{X}$ for the calculation, which leads to spurrious correlations due to the noise components. Thus PCR takes care of this and performs PCA on $\mathbf{X_{noise}}$, avoiding the issues of collinearity and noise in the input data.

Once the matrices $\mathbf{B}_{OLS}$ and $\mathbf{B}_{PCR}$ have been calculated we can evaluate their performance for in-sample and out-of-sample data to resemble $\mathbf{Y}$. We thus calculate the squared error like $|\mathbf{Y} - \mathbf{BX}|^2$, using $\mathbf{X}_{noise}$ and $\mathbf{Y}$ for in-sample evaluation, and $\mathbf{X}_{test}$ and $\mathbf{Y}_{test}$ for out-of-sample. The error is divided by the number of matrix elements, leading to the results in Table 1. These show how OLS performs better in training samples but worse in testing data, leading to the conclusion of OLS overfitting the noise present in its training data.

Method	In-sample Error	Out-of-sample Error
OLS	0.7103	0.4755
PCR	0.7154	0.4703

Table 1: Error per matrix element for OLS and PCR

In order to validate the results obtained above it is good practice to evaluate the performance of matrices $\hat{\mathbf{B}}_{OLS}$ and $\hat{\mathbf{B}}_{PCR}$ over an ensemble of test data not drawn from the same realisation as the training data. Assessing the performance of both over an ensemble of 100 test realisations as before, gives the results shown in Table 2, which confirm the results obtained above, showing PCR superiority in out-of-sample data.

Method	Out-of-sample Error
OLS	0.4799
PCR	0.4560

Table 2: Error per matrix element for OLS and PCR for 100 realisations ensemble

2 Adaptive signal processing

2.1 The Least Mean Square (LMS) Algorithm

Given the second order auto-regressive process satisfying the difference equation

$$x(n) = a_1 x(n-1) + a_2 x(n-2) + \eta(n) \tag{9}$$

Where $[a_1, a_2] = [0.1, 0.8]$, and $\eta(n) = \mathcal{N}(0, \sigma_n^2 = 0.25)$, we can approximate a_1, a_2 using the Least Mean Squares (LMS) algorithm. To evaluate convergence of the LMS algorithm a sufficient condition is $0 < \mu < 2/\lambda_{max}$. Where λ_{max} refers to the largest eigenvalue of the autocorrelation matrix $\mathbf{R_{xx}}$, where $\mathbf{x}(n) = [x(n-1), x(n-2)]^T$.

$$\mathbf{R_{xx}} = \mathbb{E}\{[x(n-1), x(n-2)]^T [x(n-1), x(n-2)]\} = \mathbb{E}\begin{bmatrix} x(n-1)x(n-1) & x(n-1)x(n-2) \\ x(n-2)x(n-1) & x(n-2)x(n-2) \end{bmatrix} \tag{10}$$

If Equation 9 is multiplied by $x(n-k)$, all values of $\mathbf{R_{xx}}$ can be obtained by taking the expectation.

$$\mathbb{E}\{x(n)x(n-k)\} = \mathbb{E}\{a_1 x(n-1)x(n-k)\} + \mathbb{E}\{a_2 x(n-2)x(n-k)\} + \mathbb{E}\{\eta(n)x(n-k)\} \tag{11}$$

Defining the autocorrelation function $r(k) = \mathbb{E}\{x(n)x(n-k)\}$, and considering $\mathbb{E}\{\eta(n)x(n-k)\}$ vanishes for $k > 0$, we can vary $k = 0, 1, 2$ to obtain the system of equations below from Equation 11.

$$\begin{aligned} r(0) &= a_1 r(1) + a_2 r(2) + \sigma_n^2 \\ r(1) &= a_1 r(0) + a_2 r(1) \\ r(2) &= a_1 r(1) + a_2 r(0) \end{aligned} \tag{12}$$

From these we can solve for r(k), to get

$$\mathbf{R_{xx}} = \begin{bmatrix} r(0) & r(1) \\ r(1) & r(0) \end{bmatrix} = \begin{bmatrix} 25/27 & 25/54 \\ 25/54 & 25/27 \end{bmatrix} \tag{13}$$

Hence by performing eigenvalue decomposition on $\mathbf{R_{xx}}$ the largest eigenvalue is obtained, $\lambda_{max} = 1.3889$. From this convergence of the LMS algorithms can be satisfied by setting $0 < \mu < 2/\lambda_{max} = 1.44$ to the Wiener optimal solution.

The signal in Equation 9 is implemented using 1000 random samples, and the LMS algorithm is developed to estimate the process' coefficients. By varying the step size $\mu = 0.01, 0.05$ we can evaluate the performance of the algorithm using the squared error with the actual coefficients, Figure 13 (left).

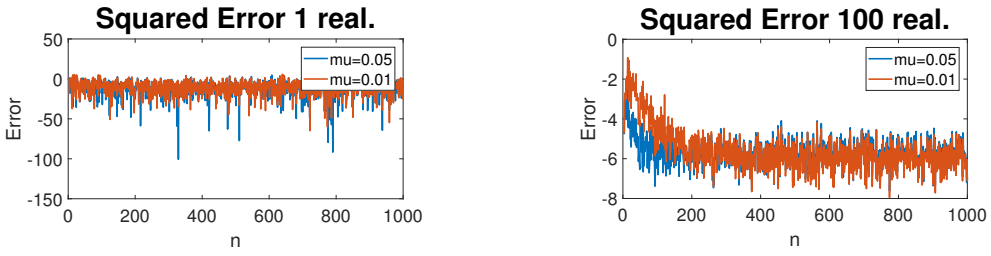


Figure 13: Squared error evolution using LMS algorithm

Performing a single realisation makes evaluation hard, as both values of μ oscillate around a steady state and make direct comparison difficult. An average error signal is obtained from an ensemble of 100 realisations from which a much better evaluation can be performed, Figure 13(right). It is observed how the larger step size converges faster, however with time reaches a higher steady state error. This matches what we would expect, the larger step sizes makes larger updates towards the solution however fails to properly reach to due to its bigger step size.

The fluctuations in steady state solution of the LMS introduce excess error. This excess error can be calculated as the Excess Mean Squared Error (EMSE), as the difference between the MSE and the minimum attainable MSE (σ_n^2). The corresponding misadjustment, $\mathcal{M}$, is then the ratio of this excess error and σ_n^2, $\frac{EMSE}{\sigma_n^2}$, which for small step sizes can be approximated as $\frac{\mu}{2}\text{Tr}\{\mathbf{R_{xx}}\}$. From the ensemble the steady state can be calculated to get needed EMSE and then calculate $\mathcal{M}$. Table 3, shows the theoretical and calculated values of $\mathcal{M}$, showing how as discussed even though a smaller step size might converge slower, leads to less misadjustment as it reduces oscillations.

Stepsize μ	$\mathcal{M}_{estimates}$	$\mathcal{M}_{theoretical}$
0.01	0.0104	0.0093
0.05	0.0511	0.0463

Table 3: Theoretical and Calculated misadjustment for $\mu = 0.01, 0.05$

Figure 14 shows the evolution of the estimated coefficients for 1 and 100 realisations for $\mu = 0.01, 0.05$. it is clear how oscillations affect clearly the result, and to see clear evolution the ensemble average is needed to perform a comparison. As seen before, a larger step size results in a quicker convergence, however Figure 14 shows clearly how smaller step size actually leads to better estimation. Even though the estimates for a_1 are correct, a_2 estimates are not as accurate, specially for larger step size. From the single realisation graph we can see how smaller step size gives much more stable coefficient evolution, whilst the ensemble graph shows a clear final results comparison of coefficient estimation.

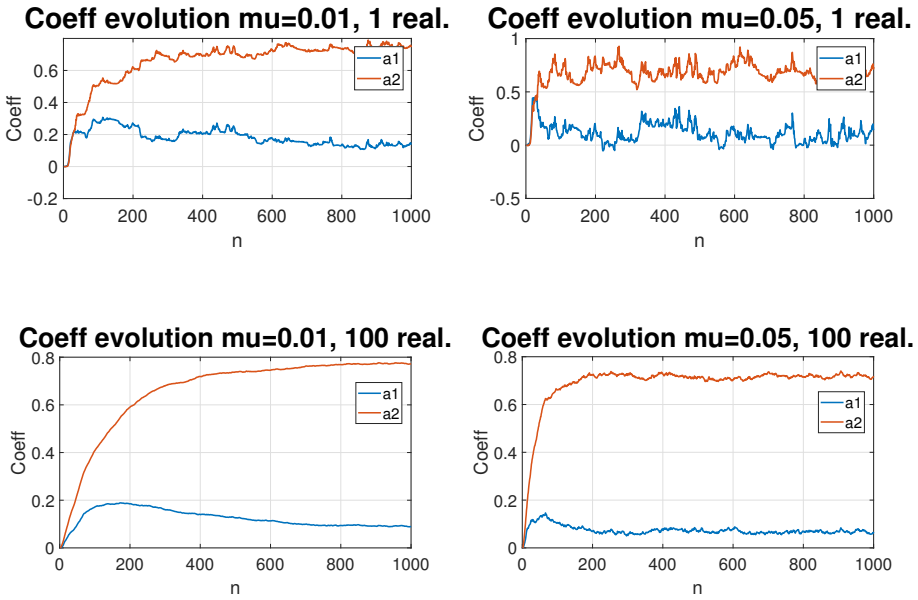


Figure 14: Coefficient evolution for varying step size and realisations

The leaky LMS algorithm introduces a leakage in the LMS algorithm to provide a way of convergence, when eigenvalues of the signal's autocorrelation function are zero. Its cost function is

$$J_2(n) = \frac{1}{2}(e^2(n) + \gamma||\mathbf{w}(n)||_2^2 \tag{14}$$

where $e(n) = y(n) - \mathbf{w}(n)^T\mathbf{x}(n)$.The update function that minimises the above equation will be of form, Equation 15 accounting for the gradient of the cost function.

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mu\nabla_{\mathbf{w}}J_2 \tag{15}$$

Deriving the gradient we get

$$\begin{aligned}\nabla_{\mathbf{w}} J_2 &= \frac{1}{2}\left(\frac{\delta(y-\mathbf{w}^T\mathbf{x})^2}{\delta \mathbf{w}} + \gamma \frac{\delta \mathbf{w}^T\mathbf{w}}{\delta \mathbf{w}}\right) \\ &= \frac{1}{2}(-2\mathbf{x}(y-\mathbf{w}^T\mathbf{x}) + 2\gamma\mathbf{w}) \\ &= -e\mathbf{x} + \gamma\mathbf{w}\end{aligned} \tag{16}$$

Substituting into Equation 15 the leaky LMS equation desired is obtained:

$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) - \mu(-e(n)\mathbf{x}(n) + \gamma\mathbf{w}(n)) \\ &= (1-\mu\gamma)\mathbf{w}(n) + \mu e(n)\mathbf{x}(n)\end{aligned} \tag{17}$$

The LMS algorithm is a weight update method which converges to the Wiener-Hopf which minimises squared error, $\mathbf{w}_{opt}$. This solution is the equivalent of the solution to $\mathbf{R}_{xx}\mathbf{w}opt = \mathbf{x}$, where $\mathbf{R}_{xx}$ is the signal's autocorrelation matrix.

The leaky LMS algorithm is introduced when the singularity of $\mathbf{R}_{xx}$ makes it non-invertible and so $\mathbf{w}_{opt}$ is not achievable. The leaky LMS then introduces a regularisation parameters γ which deals with the unstable solution of the LMS, leading to $\mathbf{w}_{leakyLMS}$. These weights however do not correspond to the actual coefficients as $\mathbf{w}_{leakyLMS} \neq \mathbf{w}_{opt}$, hence the larger the value of γ the larger the error will be as the weights will be affected most. In order to achieve stability as well as produce the best coefficient estimation, the value of γ should be kept to a minimum.

Figure 15 shows the different coefficient evolution for varying μ and γ. It can be seen how the converged coefficients are inaccurate and how a larger γ affects negatively the estimation. The effect γ has on magnitude, Equation 17, shows how the 2^{nd} coefficient will be most affected due to the larger magnitude ($0.8 > 0.1$), which is confirmed from the plots.

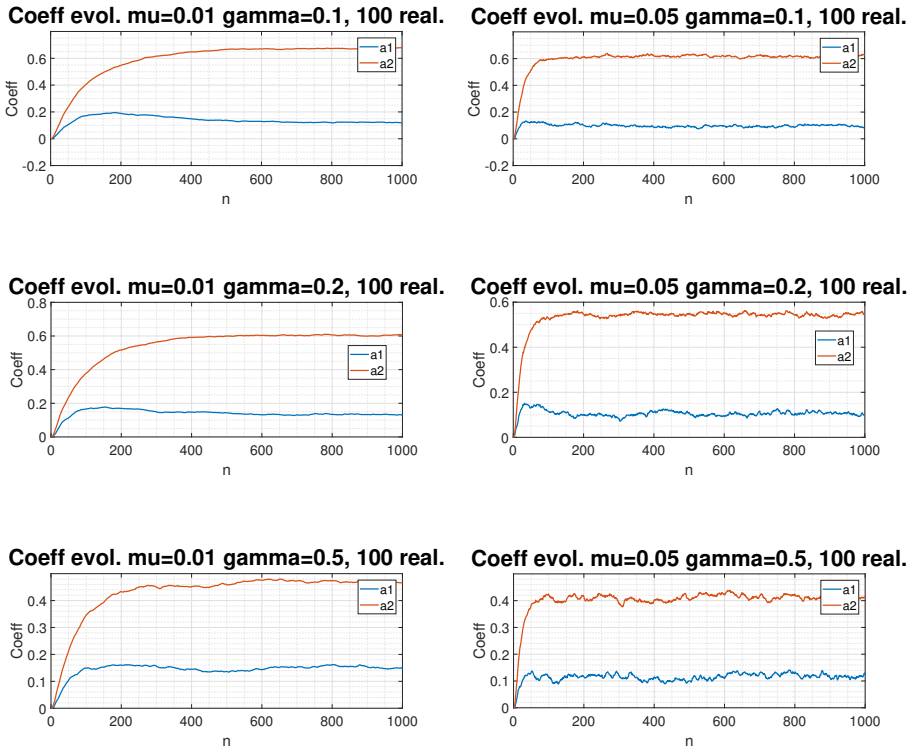


Figure 15: Coefficient evolution for varying step size μ and γ

2.2 Adaptive Step Sizes

As has been shown above a fixed step size poses different constraints on optimisation algorithms. Fixed step size introduces a trade-off between convergence speed and steady state accuracy, the

restriction these pose on algorithms mean fixed step sizes is only appropriate for a subset of signals. Ideally the step size should be large at first, thus making the first steps towards the *neighbourhood* of the optimal solution, and then reduce as the estimation approaches the solution in order to accurately update towards the optimal solution, thus reducing steady state error in lower convergence time. Variable step-size (VSS) algorithms aim to solve this, minimising the cost function with respect to the weights as well as step size, hence adapting step size to how good teh estimation is.

Gradient adaptive step-size (GASS) algorithms aim to minimise $\nabla_\mu J$ using the update rule as

$$\mu(n+1) = \mu(n) + \rho e(n)\mathbf{x}^T(n)\psi(n) \tag{18}$$

Where ψ can be set differently:

$$\begin{aligned} Beneviste &: \psi(n) = [\mathbf{I} - \mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)]\psi(n-1) + e(n-1)\mathbf{x}(n-1) \\ Ang\ \&\ Farhang &: \psi(n) = \alpha\psi(n-1) + e(n-1)\mathbf{x}(n-1), \qquad 0 < \alpha < 1 \\ Matthews\ \&\ Xie &: \psi(n) = e(n-1)\mathbf{x}(n-1) \end{aligned} \tag{19}$$

Beneviste's is essentially an adaptive low-pass filter with $e(n-1)\mathbf{x}(n-1)$ as input, which allows it to deal against noisy instantaneous gradients. ψ thus controls the gradient estimate in the update, dealing with noisy gradients. Ang & Farhang simplify this by using a simple non-adaptive low-pass filter substituting the component multiplying $\psi(n-1)$ with α, hence performing much faster at the cost of reduced accuracy in the estimations. Matthews & Xie take this a step further and remove the low-pass filter altogether, thus achieving the fastest algorithm in exchange for lower accuracy.

$$x(n) = 0.9\eta(n-1) + \eta(n) \qquad \eta \sim \mathcal{N}(0, 0.5) \tag{20}$$

All 3 methods were implemented and compared against two fixed step-size LMS algorithms in identifying a real-valued MA(1) system, Equation 20. Using $\rho = 0.005$ and $\alpha = 0.8$ Figure 15 shows the weight error $(w_0 - w(n))$ and squared error curves for the different implementations. As can be seen Beneviste performs the best, converging to the solution the fastest. Ang & Farhang follows as second best, followed from the rest. As can be seen the adaptive step-size algorithms, although more computationally expensive achieve a very low squared error and converge to the correct weights in lower iterations than the previous non-adaptive LMS algorithms.

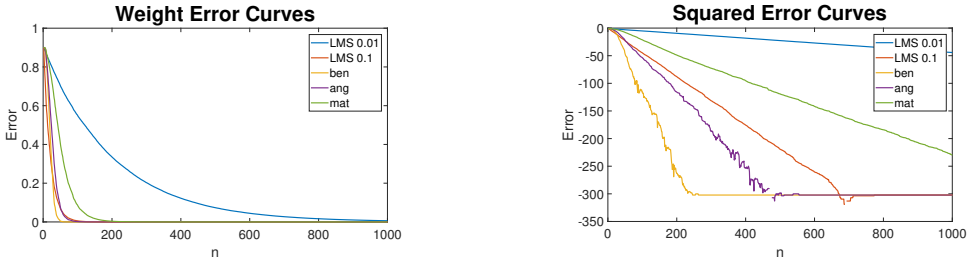


Figure 15: Weight error and Squared error evolution for different algorithms

By normalising the input signal $\mathbf{x}(\mathbf{n})$ we obtain the Normalised Least Mean Squares (NLMS), which deals with instability when $||\mathbf{x}(n)||^2$ is close to zero, by introducing a regularisation factor to become the update rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\beta}{\epsilon + ||\mathbf{x}(n)||^2} e(n)\mathbf{x}(n) \tag{21}$$

This can be verified to be an update equation starting from the update equation and error.

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \\ e_p(n) &= d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1) \end{aligned} \tag{22}$$

Substituting the first one into the second we get:

$$\begin{aligned} e_p(n) &= d(n) - \mathbf{x}^T(n)[\mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n)] \\ &= d(n) - \mathbf{x}^T(n)\mathbf{w}(n) - \mu e_p(n)||\mathbf{x}(n)||^2 \\ &= \frac{d(n) - \mathbf{x}^T(n)\mathbf{w}(n)}{1 + \mu||\mathbf{x}(n)||^2} \\ &= \frac{e(n)}{1 + \mu||\mathbf{x}(n)||^2} \end{aligned} \tag{23}$$

Substituting back into the update equation:

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \\ &= \mathbf{w}(n) + \mu \frac{e(n)}{1 + \mu||\mathbf{x}(n)||^2} e(n)\mathbf{x}(n) \\ &= \mathbf{w}(n) + \frac{1}{\frac{1}{\mu} + ||\mathbf{x}(n)||^2} e(n)\mathbf{x}(n) \end{aligned} \tag{24}$$

From which we can confirm the NLMS is follows an update equation and identify that $\epsilon = 1/\mu$ and $\beta = 1$.

The Generalised Normalised Gradient Descent (GNGD) algorithm improves stability further by making ϵ time variant and thus adaptive. *epsilon* is thus updated according to gradient as

$$\epsilon(n+1) = \epsilon(n) - \rho\mu \frac{e(n)e(n-1)\mathbf{x}^T(n)\mathbf{x}(n-1)}{(\epsilon(n-1) + ||\mathbf{x}(n-1)||^2)^2} \tag{25}$$

with $\epsilon(0)$ initialised to $1/\mu$ as for the derivation above.

The GNGD algorithm was implemented with varying ρ and μ to compare against the Beneviste implementation, in identifying the MA(1) system, Equation 20. Figure 16 shows how for all varying parameters GNGD achieves the optimal solution, just at varying speeds.

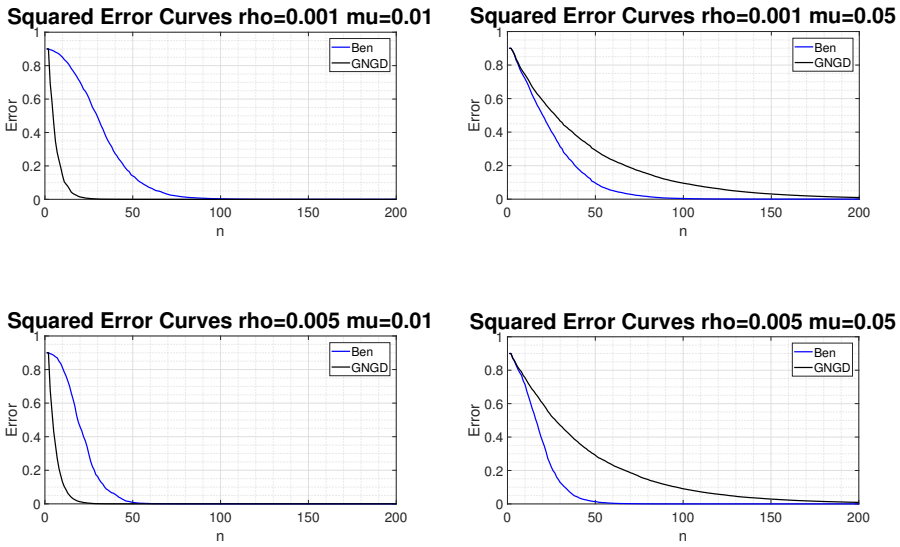


Figure 16: Weight error for varying μ and ρ of GNGD algorithm

Comparing the speed of convergence we can see how GNGD can be much faster if initialised properly. The Beneviste implementation does not vary significantly in convergence speed for different parameters and achieved the optimal solution in less than 100 iterations. The GNGD algorithm does however vary according to the parameter values. If we set μ low, the update becomes much better and GNGD does achieve the optimal solution much faster than Beneviste, without suffering from instability.

In terms of complexity, tables 4 and 5 show the breakdown of the complexity of both algorithms according to the number of multiplications and additions at each step. From this analysis we can see how the GNGD algorithm scales better than the Beneviste implementation with linear complexity ($O(M)$) against Beneviste's squared complexity ($O(M^2)$), where M denotes the number of elements in w.

Step	Action	Multiplications	Additions
$e(n)$	$y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$	M	M-1
	$d(n) - y(n)$	1	1
$\psi(n)$	$A = e(n-1)x(n-1)$	M	0
	$B = \mathbf{I} - \mu(n-1)x(n-1)x^T(n-1)$	M^2+M	M
	$A + B \times \psi(n-1)$	M^2	M^2
$\mu(n-1)$	$C = x^T(n)\psi(n)$	M	M-1
	$\mu(n) + (\rho e(n) \times C)$	2	1
$w(n-1)$	$w(n) + \mu e(n)x(n)$	M+1	M
Sum		$2M^2$+5M+3	M^2+4M
Complexity		$O(M^2)$	$O(M^2)$

Table 4: Beneviste complexity

Step	Action	Multiplications	Additions
$e(n)$	$y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$	M	M-1
	$d(n) - y(n)$	1	1
$\epsilon(n-1)$	$A = x^t(n)x(n-1)$	M	M-1
	$B = A \times e(n)e(n-1)$	2	0
	$C = (\epsilon(n-1) + \|\|x(n-1)\|\|^2)^2$	M+1	M
	$\epsilon(n) - \rho\mu(B/C)$	3	1
$w(n+1)$	$D = \|\|x(n)\|\|^2$	M	M-1
	$E = \frac{\beta e(n)}{\epsilon(n)+D}$	2	1
	$w(n) + (E \times x(n))$	M	M
Sum		5M+8	5M
Complexity		$O(M)$	$O(M)$

Table 5: GNGD complexity

2.3 Adaptive Noise Cancellation

This section deals with two denoising adaptive filter configurations, Adaptive Line Enhancer (ALE) and Adaptive Noise Cancellation (ANC). For both noise suppression configurations, estimate of the mean square prediction error is the difference between the clean signal $x(n)$ and the estimated denoised signal $\hat{x}(n)$.

$$MSPE = \frac{1}{N}\sum_{n=0}^{N-1}(x(n) - \hat{x}(n))^2 \tag{26}$$

In ALE we receive a signal $s(n) = x(n) + \eta(n)$, where $x(n)$ is the desired signal and $\eta(n)$ is coloured noise by filtering $v(n) \sim \mathcal{N}(0,1)$ as $\eta(n) = v(n) + 0.5v(n-2)$. As such the denoised signal is calculated as $\hat{\mathbf{x}}(n) = \mathbf{w}^T(n)\mathbf{u}(n)$ where $\mathbf{u}(n) = [s(n-\Delta), \ldots, s(n-\Delta-M+1)]^T$.

The ALE algorithm will thus minimise the MSE between clean and estimated signal

$$\begin{aligned}\mathbb{E}\{(s(n) - \hat{x}(n))^2\} &= \mathbb{E}\{(\eta(n) + x(n) + \hat{x}(n))^2\} \\ &= \mathbb{E}\{\eta^2(n)\} + \mathbb{E}\{(x(n) - \hat{x}(n))^2\} + 2\mathbb{E}\{\eta(n)(x(n) - \hat{x}(N))\} \\ &= \mathbb{E}\{\eta^2(n)\} + \mathbb{E}\{(x(n) - \hat{x}(n))^2\} - 2\mathbb{E}\{\eta(n)\hat{x}(n)\}\end{aligned} \tag{27}$$

From expanding the MSE we observe how three components can be considered. The first term, the noise power is independent of Δ and so is of little use, the second term is delat with by LMS

procedure and is independent of noise, hence not really relevant. Finally the last term is dependent on Δ via $\hat{x}(n)$ and so is the term to consider for further examination. This term would ideally equal zero signifying η and $\hat{x}$ beign uncorrelated.

$$\begin{aligned}\mathbb{E}\{\eta(n)\hat{x}(n)\} &= \mathbb{E}\left\{(v(n)+0.5v(n-2))\mathbf{w}^T(n)\mathbf{u}(n)\right\} \\ &= \mathbb{E}\left\{(v(n)+0.5v(n-2))\left(\sum_{i=0}^{M} w_i(x(n-\Delta-i)+\eta(n-\Delta-i))\right)\right\} \\ &\text{Using the fact that } x \ \& \ v \text{ are uncorrelated.} \\ &= \mathbb{E}\left\{(v(n)+0.5v(n-2))\left(\sum_{i=0}^{M} w_i\eta(n-\Delta-i)\right)\right\} \\ &= \mathbb{E}\left\{(v(n)+0.5v(n-2))\left(\sum_{i=0}^{M} w_i(v(n-\Delta-i)+0.5v(n-\Delta-i-2)\right)\right\}\end{aligned} \tag{28}$$

From this result it is clear how for values $\Delta > 2$ the results becomes the expectation of different noise samples which are uncorrelated. Thus for a filter length of M¿1, the ALE can be used as long as $\Delta > 2$. The ALE algorithm was implemented to confirm the empirical results. Figure 17 backs up the result obtained, showing how the realisations of the signal improve considerably after for $\Delta > 2$, which is as well observed in the lower variance experienced by the ensemble average plots.

Additionally calculating the MSPE for each value of *Delta*, we see how the error decreses substantially as $\Delta > 2$. For $\Delta = 2, 3, 4$ the measured MSPE is $0.4497, 0.3077$ and 0.3097 respectively.

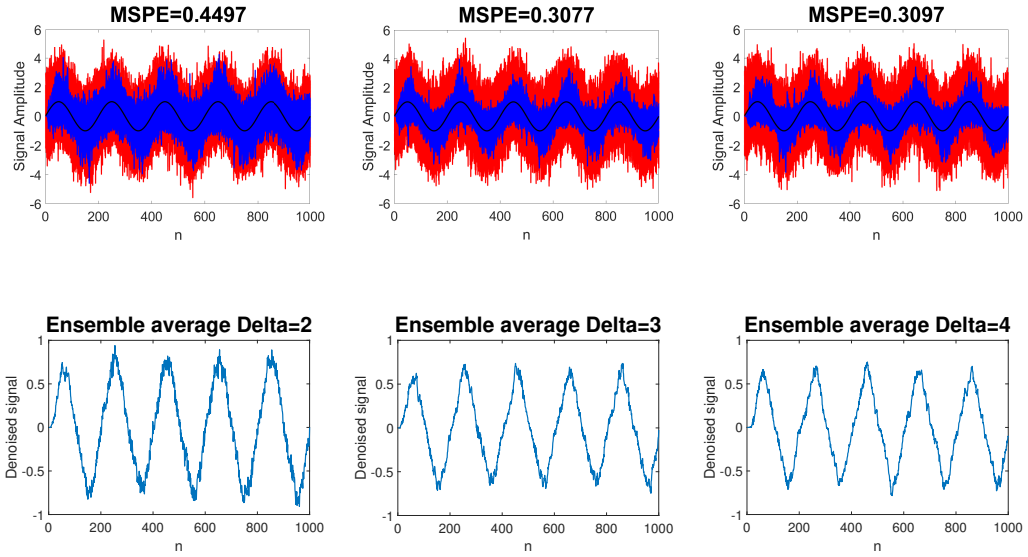


Figure 17: ALE implemented for 100 realisations for $\Delta = 2, 3, 4$ (red:noisy, blue:denoised, black:original), and corresponding ensemble averages

The values of Δ and M are varied to examine their effect on MSPE. Figure 18 shows on the right how MSPE varies for different model order as Δ is varied between 1 and 25. For all model orders we can see how MSPE drops for $\Delta = 3$ backing up the previous findings. As Δ is increased from there so does MSPE increase steadily, this is due to $\mathbf{u}$ & $\mathbf{x}$ becoming uncorrelated, thus leading to worse results even though the noise remains uncorrelated. This can be clearly seen in Figure 19a comparing original and denoised signals. Figure 18 on the right, shows how even though increasing Δ decorrelated the signals, the sinusoidal nature of the signal means if Δ is increased enough MSPE falls back again showing the signal's sinusoidal behaviour. From the observed plots it is clear the best pragmatic choice would be $\Delta = 3$.

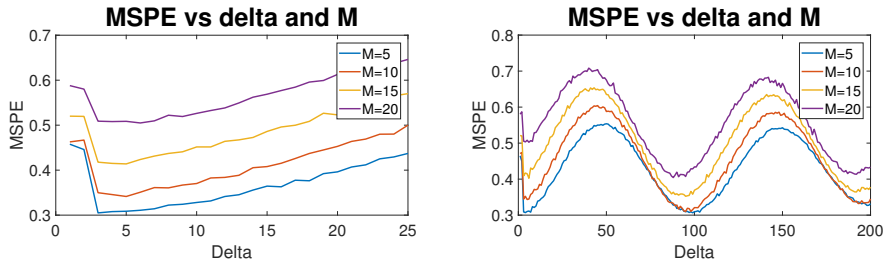
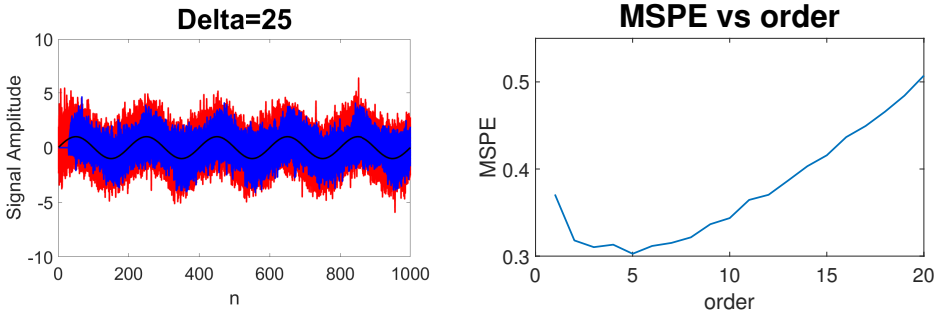


Figure 18: MSPE for varying Δ and model order

Model order also affects the performance of the algorithm at denoising the signal. As can be seen from Figures 18 and 19b the minimum error is obtained for M=5 and MSPE increases with model order. This signifies the overfitting of the model, as model order increases the algorithm becomes more complex and aims at modelling the noise (overfitting) which results in higher error. Considering complexity increases with model order it is clear we would aim to keep it low, from which the best pragmatic choice would be M=5.



(a) 100 ALE realisations for $\Delta = 25$

(b) MSPE vs model order

We now implement the Adaptive Noise Cancellation (ANC) algorithm, which has as inputs the corrupted noise $s(n)$ and a secondary signal $\epsilon(n)$ which is somehow correlated to the noise. We thus develop a linear adaptive filter $\hat{\eta}(n) = \mathbf{w}^T(n)\mathbf{u}(n)$ with a delayed version of the noise $\mathbf{u}(n) = [\eta(n), \ldots, \eta(n-M+1)]$,with are aim to perform $\hat{x}(n) = s(n) - \hat{\eta}(n)$. As we have an additional noise correlated signal the ANC should outperform the ALE by providing a more accurate estimate of the noise.

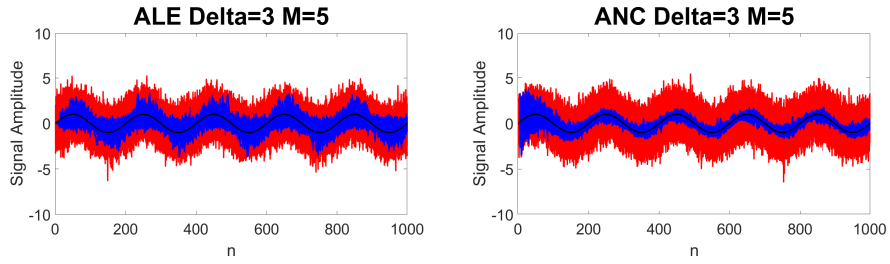


Figure 20: MSPE for varying Δ and model order

The figures above show how ANC performs significantly better at the denoising task than ALE for the given signal. The denoised realisations show a much cleaner version with a considerable difference to the original corrupted signals. The presence of the correlated secondary noise allows for the system to estimate the noise very accurately and so the denoising task is performed succesfully. The main difference between ANC and ALE is the time consideration, ALE performs similar through time, whilst ANC performs badly at first, until the weights are updated accordingly and a correct estimate of the noise is obtained. Thus if the denoising can wait for a reasonable amount

of time the ANC outperforms the ALE at denoising the signal, although if the wait were not an option ALE would have a comparable performance with ANC. The MSPEs for ALE and ANC respectively are 0.3077 and 0.0768, showing how ANC outperforms ALE.

The ANC algorithm is now performed on real data, more specifically the POz EEG signal from Section 1.2. Aiming to reduce the 50Hz artefact present, a secondary signal can be generated as a sinusoid frequency 50Hz corrupted by noise to use in the ANC algorithm.

To evaluate the performance of the algorithm a spectogram is used for visualisation using a Hamming window of length $L = 4096$ and 75% overlap. Figure 21 shows clearly the 50Hz component in the signal which we aim to remove, as well as slightly the SSVEP and its harmonics.

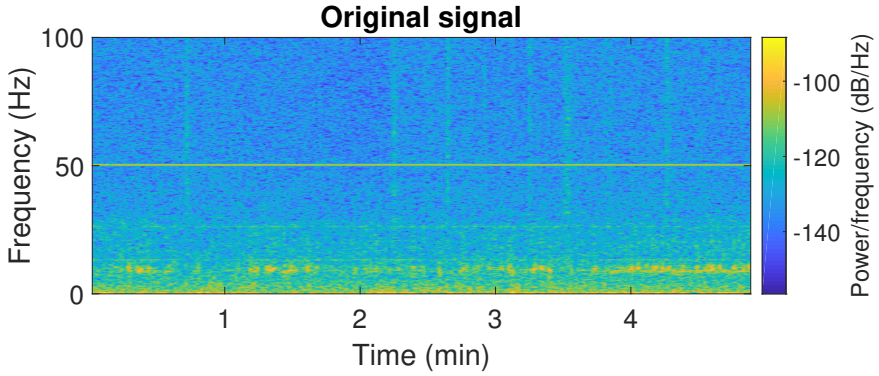


Figure 21: Spectogram of original signal

First the model order is varied to evaluate the appropriate setting. Figures 22 and 23 show the effect of model order on the denoising of the 50Hz component. Lower model orders do not completely manage to remove the 50Hz component and need to increase to $M = 20$ for satisfactory suppression. Higher model orders on the other hand do suppress the 50Hz component, but if increased too much start distorting the spectogram and the remaining components (eg, SSVEP and harmonics) become hard to identify. This is further seen in the periodogram, Figure 23, where the 50Hz component is clearly further reduced as model order increases. However from this plot we can again see how model order distorts the signal, and the actual peaks corresponding to the SSVEP are much less clear for larger valued of M. As a result if the aim of the denoising was to avoid confusion of SSVEP peaks with external interference a model order above 20 would best suit, avoiding the reduction in peak magnitude of the actual SSVEP components whilst reducing the 50Hz component so that it doesn't corrupt.

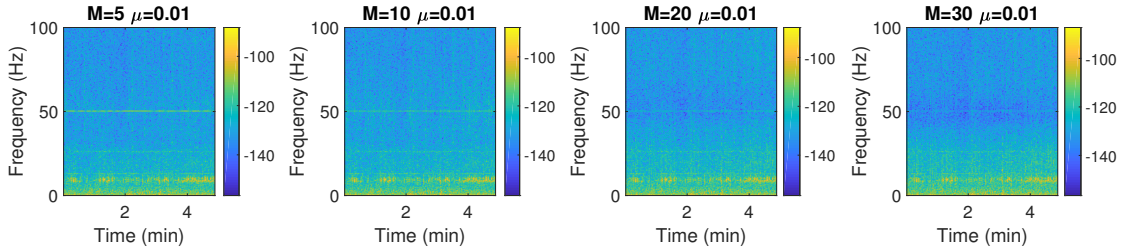


Figure 22: Signal spectogram for varying model orders

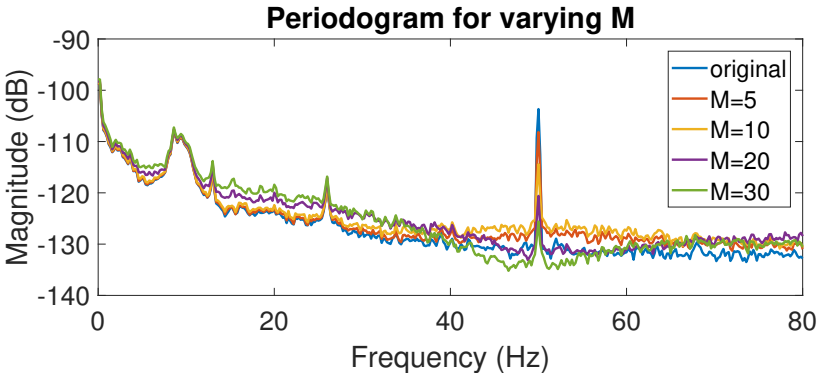


Figure 23: Period of signal for varying model orders

Varying μ will also have a considerable effect on the denoising as the update of the weights is dependent on it. Figures 24 and 25 show the results of the suppression for varying μ. It is clear how a low μ has the ability to suppress the 50Hz component succesfully without affecting the signal. Larger value however do not suppress the component as good and additionally distort the signal. From observations it is clear a larger step size prevents the algorithms from arriving at the correct solution, whilst the smaller step size is preferred as it is able to update more precisely hence achieving the best weights for denoising. As a result the optimum parameters would be M=20 and $\mu = 0.001$.

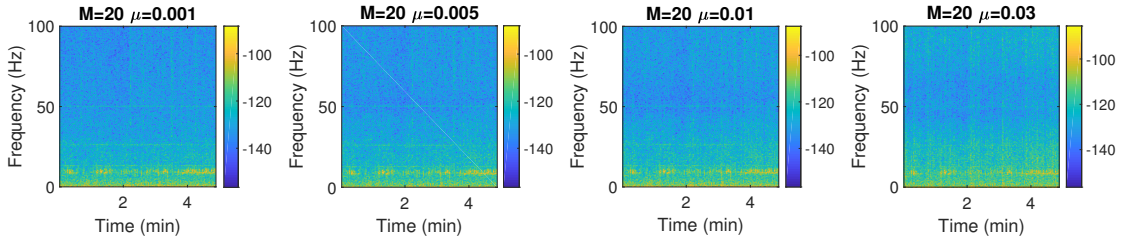


Figure 24: Signal spectogram for varying μ

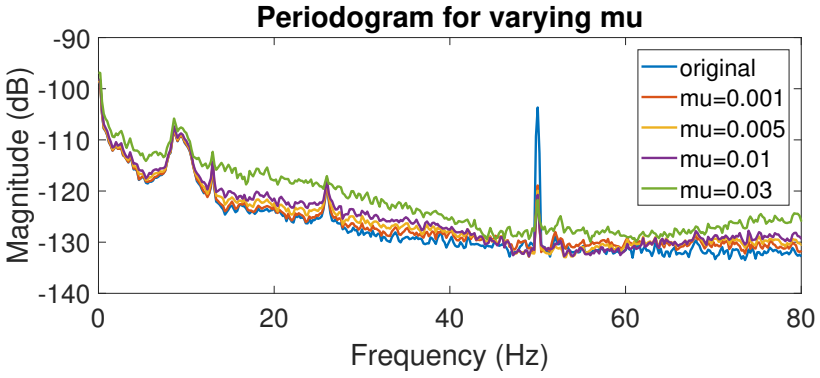


Figure 25: Period of signal for varying μ

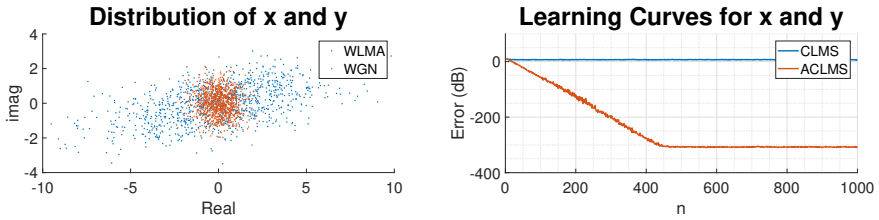
3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

Coefficient estimation of complex signals poses a new challenge by introducing the need of circularity of the system for correct estimation. By developing a Widely Linear Moving Average WLMA(1) model of form

$$y(n) = x(n) + b_1 x(n-1) + b_2 x*(n-1) \qquad x \sim \mathcal{N}(0,1) \tag{29}$$

where x is circular white noise and $b_1 = 1.5 + j, b_2 = 2.5 - 0.5j$. Figure 26a shows the distribution of both signal, showing how WGN is circular but the WLMA model is not.



(a) Distribution of circular WGN and WLMA(1)

(b) Learning curves of circular WGN and WLMA(1)

Figure 26

The Complex LMS (CLMS) and the Augmented CLMS (ACLMS) are performed on the y to examine the estimation capabilities of both algorithms. Figure 26b shows the learning curves of both algorithms, showing how CLMS fails to correctly estimate the coefficients whilst ACLMS does perform the task successfully. Without the presence of a conjugate factor CLMS is unable to correctly identify the coeffcients and so settles at a steady state error of $-7dB$, whilst the ACLMS does manage and settles at an error of $-300dB$ proving it has estimated the coefficients correctly. From this we can clearly see how CLMS is unable to estimate coefficients if the signal is non-circular, whilst the increased capabilities of the ACLMS help it deal with this situations and perform correct estimation.

We now load the bivariate wind data and develop the complex signal $v[n] = v_{east}[n] + jv_{north}[n]$ for the three regimes: low, medium and high wind. Figure 27 shows the circularity plots for the three regimes. The circularity coefficient $\rho = \frac{|\mathbb{E}\{zz^*\}|}{\mathbb{E}\{zz^T\}}$, calculates the circularity of a given signal as the ratio of covariance and pseudovariance, with a lower value indicating more circularity.

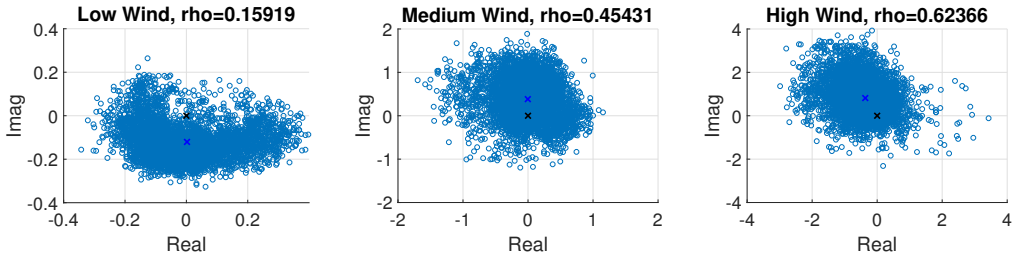


Figure 27: Circularity plots. Black ×: origin, Blue ×: mean

The circularity coefficient measures how rotation invariant the distribution is about the origin and so is not dependent on the angle. Hence the bigger the distance between the origin and the mean the higher the circularity coefficient. The above plots show this relationship with the low wind showing the highest circularity and high wind the lowest circularity. Both the circularity coefficient and the origin-mean distance show this circularity relation as expected. As expected

higher wind will vary most and so have lower circularity, whilst low wind will vary least and have higher circularity, with medium wind in between both.

The CLMS and ACLMS algorithms are performed on the three wind regimes for one-step ahead prediction of the wind data. By varying model order and setting $\mu = 0.001$ we can evaluate the performance of both algorithms according to the MSPE for the different wind regimes. Figure 28 shows the error analysis on different wind regimes and as expected the difference of CLMS and ACLMS is less pronounced for lower circularity coefficient and shows a larger performance difference for the high wind regime which has a lower circularity.

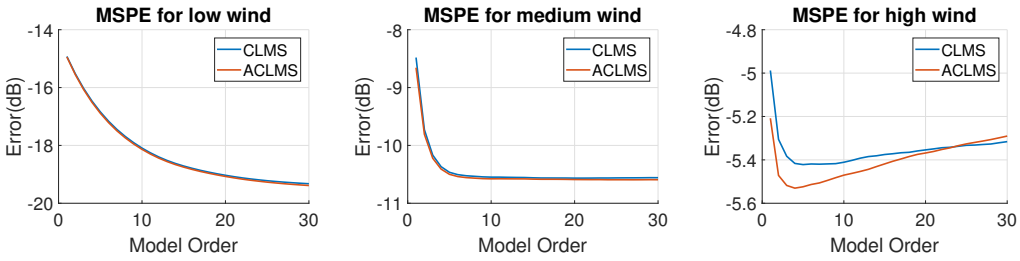


Figure 28: MSPE for varying wind regimes by CLMS and ACLMS

From the above figures we can see how circularity affects MSPE for varying model order. For low wind which is more circular, model order makes error decrease similarly for CLMS and ACLMS, whilst for medium and high wind the difference is a bit more pronounced. For all regimes the initial decrease in model order results in a decrease in error, lead by the fact that the model now has more parameters to adjust to the system. Unlike for low wind, medium and high wind seem to have a minimum error at order 4 signifying the overfitting of the data by the model. As order is increased the model starts to overfit the data and so is unable to generalise for one-step ahead prediction. This is most pronounced in the high wind plot where error starts increasing after order 4. This is clearly overfitting and the higher increase in error experienced in ACLMS in comparison to CLMS is just an indication of the increased parameters in ACLMS which lead it to overfit even more than CLMS. In order to decide model order or algorithm a penalisation measure such as AIC or MDL could be used. It is clear from the graphs that ACLMS performs best for non-circular data but has to be kept at low model order to avoid overfitting, whilst for circular data the choice seems less important and would be more to do with complexity considerations.

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

$$v(n) = A(n)e^{j(2\pi \frac{f_0}{f_s} n+\phi)} + B(n)e^{-j(2\pi \frac{f_o}{f_s} n+\phi)} \tag{30}$$

where $A(n) = \frac{\sqrt{6}}{6}(V_a(n) + V_b(n)e^{j\Delta_b} + V_c(n)e^{j\Delta_c})$ and $B(n) = \frac{\sqrt{6}}{6}(V_a(n) + V_b(n)e^{-j(\Delta_b+2\pi/3)} + V_c(n)e^{-j(\Delta_c-2\pi/3)})$. The system is balanced when $V_a(n) = V_b(n) = V_c(n)$ and $\Delta_b = \Delta_c = 0$, show in Figure 29 (left) and has a complete circular behaviour being rotational invariant. When the voltages of Δ are varied however the system becomes unbalanced and the circularity plot, figure 29, shows how the system is not longer circular. Thus the circularity diagram can be used to detect a fault in the system by measuring its circularity and checking how much this has deviated.

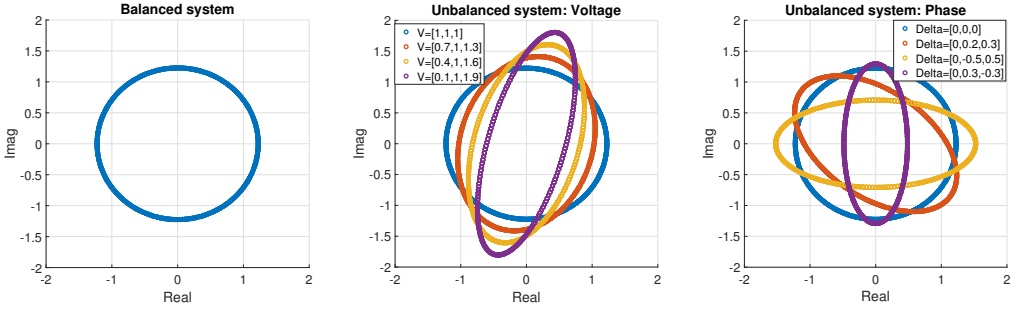


Figure 29: Circularity plots for balanced and unbalanced systems

The frequency of a balanced and unbalanced system can be computed from the Strictly and Widely Linear autoregressive models:

$$\begin{aligned} \text{Strictly Linear} &: v(n+1) = h^*(n)v(n) \\ \text{Widely Linear} &: v(n+1) = h^*(n)v(n) + g^*(n)v^*(n) \end{aligned} \tag{31}$$

For the balanced system we can express the voltage as

$$v(n) = \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \tag{32}$$

Hence by substituting from the AR model equation

$$\begin{aligned} \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s}(n+1) + \phi)} &= h^*(n) \sqrt{\frac{3}{2}} V e^{j(2\pi \frac{f_0}{f_s} n + \phi)} \\ e^{j(2\pi (f_0/f_s)} &= h^*(n) \end{aligned} \tag{33}$$

Which by expressing it in terms of real and imaginary components we can express as $h^*(n) = e^{-j \arctan(\frac{Im\{h(n)\}}{Re\{h(n)\}})}$. From which we can arrive to

$$\begin{aligned} 2\pi \frac{f_0}{f_s} &= -\arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \\ f_0 &= -\frac{f_s}{2\pi} \arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \end{aligned} \tag{34}$$

Where we can remove the negative sign since it only determines direction of rotation, to arrive to the desired result.

$$f_0 = \frac{f_s}{2\pi} \arctan\left(\frac{\Im\{h(n)\}}{\Re\{h(n)\}}\right) \tag{35}$$

For the unbalanced system, voltage is defined by Equation 30, and so by substituting to the widely linear equation we get

$$\begin{aligned} A(n)e^{j(2\pi \frac{f_0}{f_s}(n+1)+\phi)} + B(n)e^{-j(2\pi \frac{f_o}{f_s}(n+1)+\phi)} &= h^*(n)(A(n)e^{j(2\pi \frac{f_0}{f_s} n+\phi)} + B(n)e^{-j(2\pi \frac{f_o}{f_s} n+\phi)}) \\ &\quad + g^*(n)(A^*(n)e^{-j(2\pi \frac{f_0}{f_s} n+\phi)} + B^*(n)e^{j(2\pi \frac{f_o}{f_s} n+\phi)}) \end{aligned} \tag{36}$$

Rearranging with $e^{j(2\pi \frac{f_0}{f_s} n+\phi)}$ and $e^{-j(2\pi \frac{f_0}{f_s} n+\phi)}$ as references the relations are:

$$\begin{aligned} A(n+1)e^{j(2\pi \frac{f_0}{f_s})} &= h^*(n)A(n) + g^*(n)B^*(n) \\ B(n+1)e^{-j(2\pi \frac{f_0}{f_s})} &= h^*(n)B(n) + g^*(n)A^*(n) \end{aligned} \tag{37}$$

If we assume $v(n+1) \approx v(n)$ we can then assume $A(n+1) = A(n)$ and $B(n+1) = B(n)$. Thus we can rearrange to

$$\begin{aligned} e^{j(2\pi \frac{f_0}{f_s})} &= \frac{h^*(n)A(n) + g^*(n)B^*(n)}{A(n+1)} \approx h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \\ e^{-j(2\pi \frac{f_0}{f_s})} &= \frac{h^*(n)B(n) + g^*(n)A^*(n)}{B(n+1)} \approx h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \end{aligned} \tag{38}$$

From this we take the conjugate of the second term and equate both equations. After which by letting $z = \frac{B^*(n)}{A(n)}$ we get

$$\begin{aligned} h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} &= h(n) + g(n)\frac{A(n)}{B^*(n)} \\ h^*(n) + g^*(n)z &= h(n) + \frac{g(n)}{z} \\ h^*(n)z + g^*(n)z^2 &= h(n)z + g(n) \\ g^*(n)z^2 + (h^*(n) - h(n))z - g(n) &= 0 \end{aligned} \tag{39}$$

This is thus a 2^{nd} order equation which can be solved as

$$\begin{aligned} z &= \frac{(h(n) - h^*(n)) \pm \sqrt{(h^*(n) - h(n))^2 + 4g^*(n)g(n)}}{2g^*(n)} \\ &= \frac{2j\Im\{h(n)\} \pm \sqrt{-4\Im^2\{h(n)\} + 4|g(n)|^2}}{2g^*(n)} \\ &= j\frac{\Im\{h(n)\} \pm \sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{g^*(n)} \end{aligned} \tag{40}$$

Putting the solution back into Equation 38:

$$\begin{aligned} e^{j(2\pi\frac{f_0}{f_s})} &= h^*(n) + g^*(n)j\frac{\Im\{h(n)\} \pm \sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{g^*(n)} \\ &= h^*(n) + j\Im\{h(n)\} \pm j\sqrt{\Im^2\{h(n)\} - |g(n)|^2} \\ &= \Re\{h(n)\} \pm j\sqrt{\Im^2\{h(n)\} - |g(n)|^2} \\ &= |C|e^{j\left(\arctan\left(\frac{\pm\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right)\right)} \end{aligned} \tag{41}$$

Setting $|C| = 1$ since $|e^{j(2\pi(f_0/f_s))}| = 1$ we can equate the exponentials to arrive at the final solution by choosing the positive frequency

$$\begin{aligned} 2\pi\frac{f_0}{f_s} &= \arctan\left(\frac{\pm\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right) \\ f_0 &= \pm\frac{f_s}{2\pi}\arctan\left(\frac{\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right) \\ f_0 &= \frac{f_s}{2\pi}\arctan\left(\frac{\sqrt{\Im^2\{h(n)\} - |g(n)|^2}}{\Re\{h(n)\}}\right) \end{aligned} \tag{42}$$

From the above results we can estimate the $\alpha - \beta$ voltages from before from the coefficients of a Strictly or Widely-Linear AR(1) model. We thus set $f_0 = 50, f_s = 5000, \mu = 0.01, \phi = 0$ and use the previously defined CLMS and ACLMS for frequency estimation.

The estimation of the balanced system results in both algorithms converging to the correct value of 50Hz, Figure 30. As seen on the left figure CLMS converges faster than ACLMS which suffers from oscillations during the estimation. These oscillations are observed to be NaN values due to the initialisation of the system. With $h(0) = 0$ we can observe how Equations 35 and 42 would divide by zero and so suffer from oscillations. This initialisation causes NaN as the first value and additionally sets the squared root in ACLMS negative, thus leading to the oscillations seen in the initial estimations of the ACLMS algorithm. To deal with this $h(0)$ is initialised to 1 in both cases leading to the estimation shown in Figure 30(right) which shows a much smoother behaviour with both algorithms converging to the correct value, 50Hz.

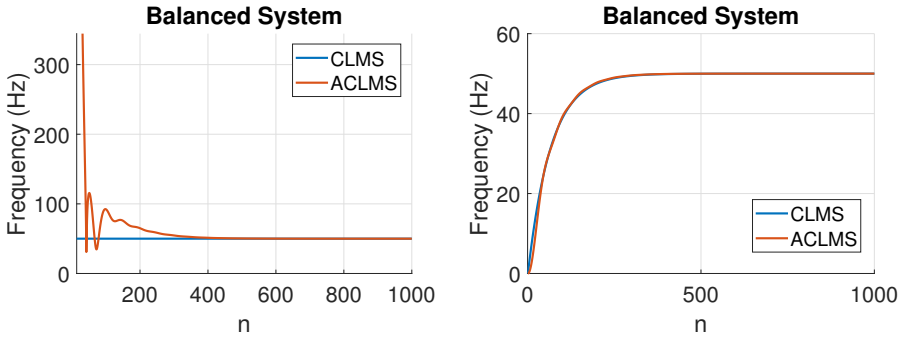


Figure 30: Frequency estimation using CLMS and ACLMS for balanced system

For an unbalanced system however CLMS and ACLMS do not perform as good. Testing the frequency estimation on both voltage and phase unbalanced system, Figure 31 shows how the estimation isn't as smooth. With the same initialisation as before for comparison we can see how ACLMS does estimate the frequency correctly and performs similar for both the balanced and unbalanced systems. CLMS however, does not succeed at estimating the correct frequency and like before fails in non-circular scenarios. The estimated frequency oscillates around a wrong value and does not converge to a single value. CLMS does not have enough degrees of freedom to estimate frequency in non-circular systems, whilst g in ACLMS does help achieve this. Although it doesn't converge at the correct value the time for convergence (to the incorrect value) of CLMS remains similar to that of the balanced system, as does ACLMS.

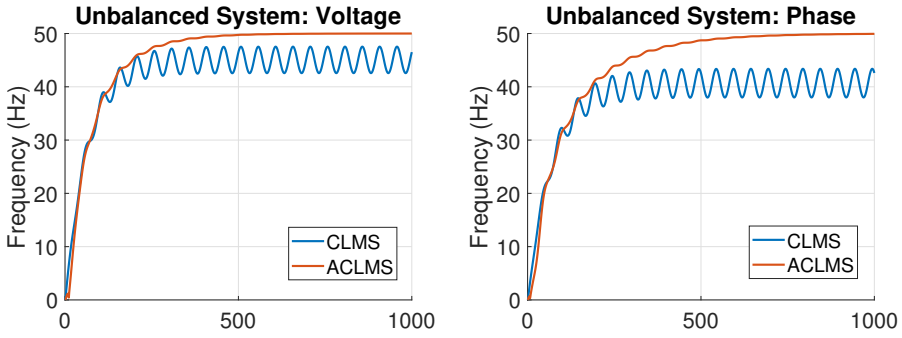


Figure 31: Frequency estimation using CLMS and ACLMS for unbalanced system

3.2 Adaptive AR Model Based Time-Frequency Estimation

We now aim to estimate the spectrum of a frequency modulated (FM) signal $y(n) = e^{j(\frac{2\pi}{f_s}\phi(n))} + \eta(n)$ where $\eta(n)$ is a circular complex-valued white noise with zero mean and variance $\sigma_\eta^2 = 0.05$. Figure 32(left) shows how the frequency varies for the signal.

To estimate the spectrum of the signal AR models are first used with `aryule` and `freqz`. Figure 32 (right) shows the estimated spectrums for different AR models. As we can see for low order models the spectrum shows little information and doesn't help in identifying the frequency components of the signal. For AR(1) in fact shows a peak which doesn't represent the constant section but the average frequency of the signal. As shown higher order models do identify the constant section peak and other components of the frequency spectrum. The system is not stationary and so the AR models cannot estimate the spectrum correctly.

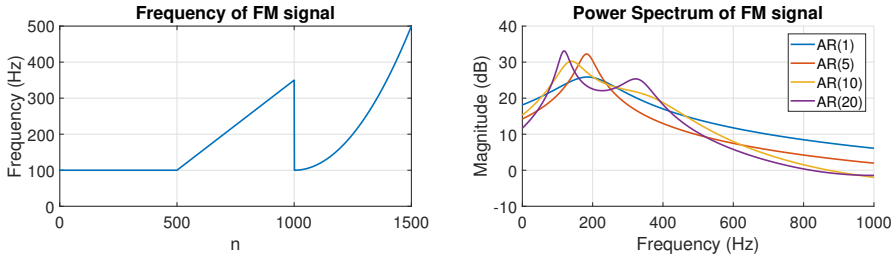


Figure 32: Frequency and AR spectrum of FM signal

As we can see from Figure 32 (left) the frequency varies in 3 clear sections: constant, linear and exponential. In order to extract the different frequency components of the signal, we can then perform spectrum estimation over the three different sections treating them as three different signals. This way we would expect to achieve the correct spectrum estimation of the signal as the sudden changes would be avoided. Figure 33 shows the different spectrum estimations for varying orders of AR models. It is clear how the constant section is correctly estimated showing a clear peak for larger orders and correctly predicted for AR(1). The linear and exponential sections

however are not as accurate and even though some extra information is present, the AR(1) does not provide information regarding the frequency spectrum of these sections.

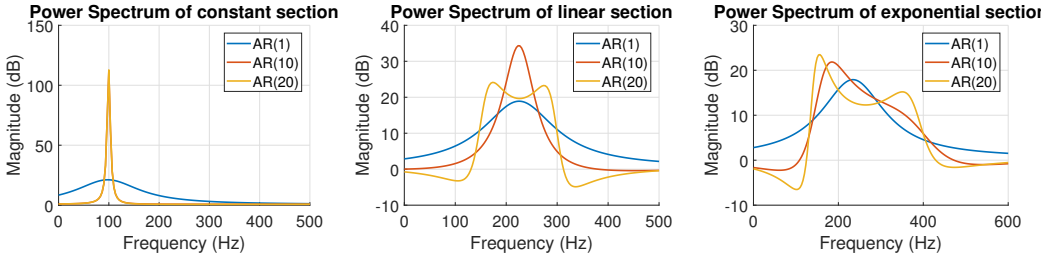


Figure 33: Frequency and AR spectrum of FM signal for different sections

Implementing the CLMS algorithm to estimate the AR coefficients of the AR model at every step to achieve a better spectrum estimation. In this case we perform the CLMS estimation at each time instant $y(n+1) = a_1^*(n)y(n)$. We use this to estiamte the time frequncy plot after removing outliers and find that the plots resemble very much that of the original frequency variation in Figure 32, as is shown in Figure 34 for varying values of μ. The first observation is the thickness of the highlighted area, which is due to the use of AR(1) and the corresponding wide main lobe as seen before. Thus if a higher model order was to be used the estimation would be more precise as the main lobe would be much narrower as seen before for AR(10) and AR(20). From varying μ what we can observe is that if mu is too small as when it equals 0.004 the learning rate is too slow and so the slow convergence means the estimation is only present for larger values of n, plus the actual estimation is not sufficiently adaptive and can't replicate the bigger changes int eh linear and exponential area, compromising the final estimation. If μ is set too high however (eg,0.3), we find that the estimation is very noisy. A higher value does however capture the instantaneous changes such as at $n = 1000$. As plotted $\mu = 0.05$ offers a good trade-off between the advantages and disadvantages of too little or too small values showing clearly the frequency variation.

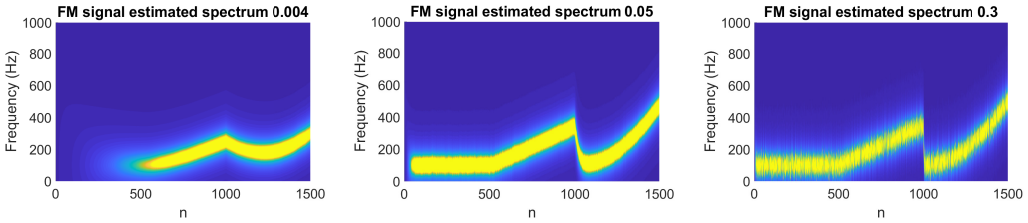


Figure 34: Time Frequency spectrum of FM signal using CLMS algorithm

3.3 A Real Time Spectrum Analyser Using Least Mean Square

Signal estimation can be done using a linear combination of N harmonically related sinusoids

$$\begin{aligned}\hat{y}(n) &= \sum_{k=0}^{N-1} w(k)e^{j2\pi kn/N} \\ &= w(0) + w(1)e^{j2\pi n/N} + w(2)e^{j(2\pi n/N)2} + \ldots + w(N-1)e^{j(2\pi n/N)(N-1)}\end{aligned} \tag{43}$$

where $\hat{y}(n)$ is the estimate signal and $w(k)$ are the unknown weights to be estimated and are the Fourier coefficients corresponding to the DFT of the signal $y(n)$. Collecting all N estimates of the signal $y(n)$ into a vector allows to express the problem as $\hat{\mathbf{y}} = \mathbf{F}\mathbf{w}$

$$\begin{bmatrix} \hat{y}(0) \\ \hat{y}(1) \\ \vdots \\ \hat{y}(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & e^{j\frac{2\pi}{N}(1)(1)} & \ldots & e^{j\frac{2\pi}{N}(1)(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & e^{j\frac{2\pi}{N}(N-1)(1)} & \ldots & e^{j\frac{2\pi}{N}(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ \vdots \\ w(N-1) \end{bmatrix} \tag{44}$$

Hence to evaluate the accuracy of the signal estimate the we aim to minimise the squared error cost function

$$\min_{\mathbf{w}} ||\mathbf{y} - \hat{\mathbf{y}}||^2 = \min_{\mathbf{w}} \sum_{n=0}^{N-1} |y(n) - \hat{y}(n)|^2 \tag{45}$$

To find the solution to the squared error function

$$\begin{aligned} ||\mathbf{y} - \hat{\mathbf{y}}||^2 &= (\mathbf{y} - \hat{\mathbf{y}})^H (\mathbf{y} - \hat{\mathbf{y}}) \\ &= (\mathbf{y} - \mathbf{F}\mathbf{w})^H (\mathbf{y} - \mathbf{F}\mathbf{w}) \\ &= \mathbf{y}^H\mathbf{y} - \mathbf{w}^H\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}\mathbf{w} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w} \\ &= \mathbf{y}^H\mathbf{y} - 2\mathbf{y}^H\mathbf{F}\mathbf{w} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w} \end{aligned} \tag{46}$$

By taking the derivative with respect to the weights and equate to zero

$$\begin{aligned} \frac{\partial(||\mathbf{y} - \hat{\mathbf{y}}||^2)}{\partial \mathbf{w}} &= -2\mathbf{F}^H\mathbf{y} + 2\mathbf{F}^H\mathbf{F}\mathbf{w} = 0 \\ \mathbf{F}^H\mathbf{F}\mathbf{w} &= \mathbf{F}^H\mathbf{y} \\ \mathbf{w} &= (\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{y} \end{aligned} \tag{47}$$

Thus the estimated signal can be found as $\hat{\mathbf{y}} = \mathbf{F}(\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H\mathbf{y}$ and the Fourier coefficients as $\mathbf{w}$.

The implication of $\mathbf{F}$ in the least squares interpretation of the DFT is that of a matrix composed of orthogonal columns which span an orthogonal subspace, and are the basis of such. The orthogonality comes from the fact that

$$\langle e^{j2\pi kn/N}, e^{j2\pi km/N} \rangle = \sum_{n=0}^{N-1} e^{j2\pi k(n-m)/N} = \begin{cases} 1 & \text{if n=m} \\ 0 & \text{otherwise} \end{cases} \tag{48}$$

As $\hat{\mathbf{y}} = \mathbf{F}\mathbf{w}$ the resulting estimation lies in the subspace spanned by the columns of $\mathbf{F}$, whilst $\mathbf{y}$ lies in another completely different subspace. The least squares solution then performs a projection operation of the real signal $\mathbf{y}$ into the subspace spanned by $\mathbf{F}$, aiming to develop the closest estimate to the real signal. The least squares solution uses the projection matrix $\mathbf{P} = \mathbf{F}(\mathbf{F}^H\mathbf{F})^{-1}\mathbf{F}^H$ on $\mathbf{y}$ to achieve $\hat{\mathbf{y}}$. The number of columns of $\mathbf{F}$, N, affects the accuracy of the estimate as the higher the number the better the estimate reaching equality when $N \to \infty$.

The DFT can be performed as a least squares solution allows for spectrum estimation using the CLMS algorithm, by estimating the Fourier coefficients on every time-step. Figure 35 shows the spectrum for the original signal. From the plot we can see how the algorithm performs successfully for the constant period of the signal and then does consider the linear and exponential sections. The issue if we observe is how the components are everlasting and do not get updated, thus producing the overlapping plot seen. CLMS being a gradient based algorithm, the Fourier weights do not get updated quick enough and so superimpose each other, this is because the updates are not done block-based and so the error backpropagation is very slow, making memory to fade slowly and not properly adapt to the new sections. This behaviour can eb observed for the first sinusoid corresponding to the constant section which doesn't fade through the other sections and instead remains significant.

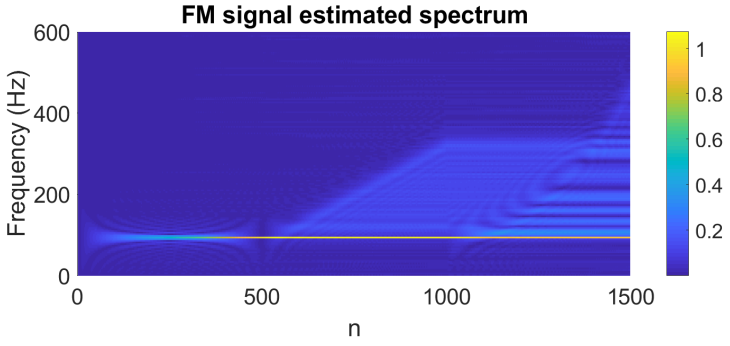


Figure 35: Time Frequency spectrum of FM signal using DFT-CLMS algorithm

In order to deal with this slow update that prevents the DFT-CLMS from producing a correct spectrum estimation, a modification of the algorithm is developed related to the work of Section 2.1 leaky LMS. In an effort to help the forgetting ability and produce a faster update the leaky CLMS is proposed, described as

$$\text{Leaky DFT CLMS}: w(n+1) = (1-\gamma\mu)w(n) + \mu e^*(n)x(n) \tag{49}$$

γ allows for the update to forget by weighing the previous weights so they have less importance. The effect of γ is explored in Figure 36. As can be seen a small γ leads to still slow update and so suffers from the same issue as before, expected for too small γ as has insufficient effect. A large γ on the other hand results in a noisy output, the fast fading causes only the last samples to eb relevant which results in a less smoother output than ideally. The best setting found was $\gamma = 0.05$ which provides a sufficient forget factor for fast updates, without distorting the signal. The spectrum estimate shown for this value is a very good estimate of the original signal frequency change and has a narrower band than using the previous CLMS method.

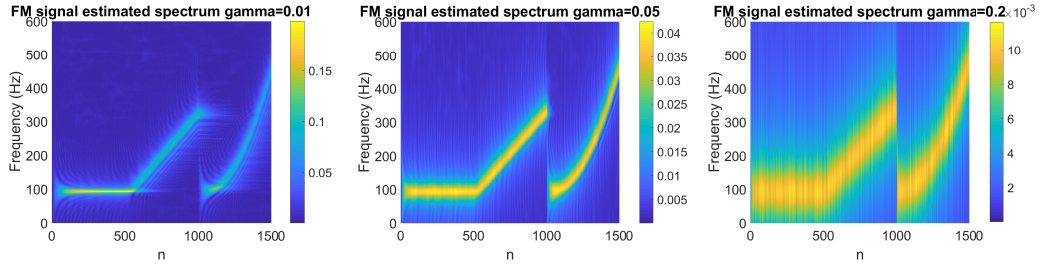


Figure 36: Time Frequency spectrum of FM signal using leaky DFT-CLMS algorithm

We finally perform the spectrum estimation on the EEG POz data from part 1.2. The stationary nature of the data means the use of leaky-CLMS is not needed and CLMS by its own is sufficient. To avoid computational burden we select the segment `36000:37200` of the signal samples for the spectrum estimation. The SSVEP signals at 13Hz and 26Hz are clearly distinguishable and the mains interference at 50Hz is also very clearly seen. The spectrum estimation hence shows that the CLMS is a good way for spectrum identification however the 3^{rd} harmonic is not visible, unlike in the original analysis where all the harmonics were visible.

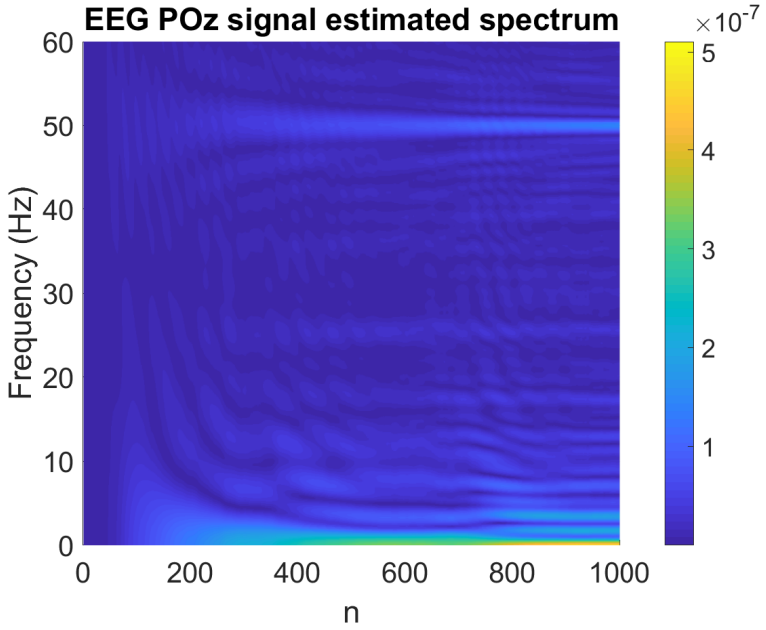


Figure 37: Time Frequency spectrum of EEG POz signal using DFT-CLMS algorithm

4 From LMS to Deep Learning

As seen in section 2 the approach towards signal prediction for non-stationary data mainly relies on the LMS algorithm. The general trend however is moving towards deep learning as a means of solving complicated classification and regression problems. In this section we will introduce the use of LMS and its variants in the aim of estimating the signal shown in Figure 38.

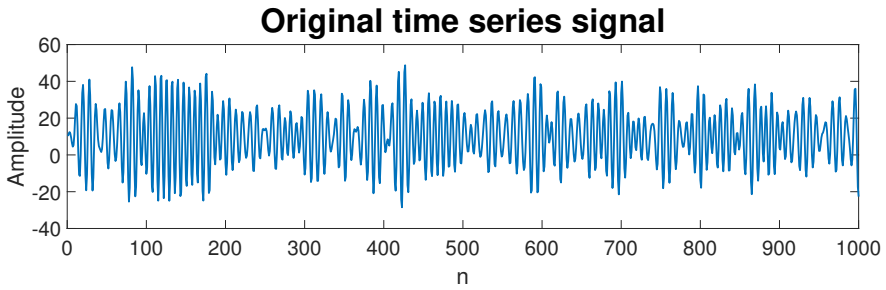


Figure 38: A non-stationary time series with non-zero mean

As we can see the signal is non-stationary and has non-zero mean. Hence the signal has its mean removed to make it zero mean and be suitable for on-step ahead prediction. The LMS algorithm is the used to achieve adaptive learning for the non-stationary signal, treating it as an AR(4) process. The one-step ahead prediction is thus produced from the 4 previous samples and shown in Figure 39. As we can see the LMS takes a few samples to converge and the prediction is far from correct initially. As LMS progresses however the prediction improves achieving a satisfactory prediction of the signal, as can be seen in the zoomed plot the prediction is quite accurate matching the original signal. The calculated $MSE = 40.0943$ shows however how there is room for improvement, whilst the prediction gain $R_p = 10\log_{10}\left(\frac{\hat{\sigma}_y^2}{\sigma_e^2}\right) = 5.1966$ dB, where $\hat{\sigma}_y^2$ is the LMS output variance and σ_e^2 the error variance, shows how the prediction error is significantly lower than the signal. The LMS by itself provides a good prediction, however fails to perfectly adjust to the original signal, Figure 39 (right).

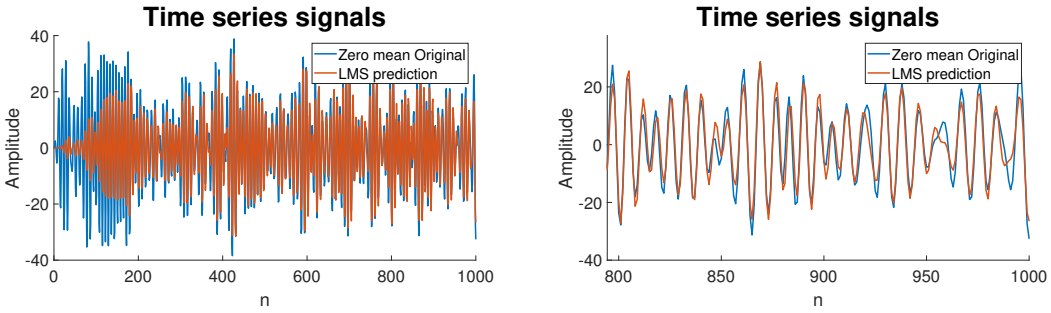


Figure 39: Original and LMS predicted time series signals. Left: complete, Right: zoomed

Typically, the generating process for the data is unknown and non-linear. Therefore to capture this we can add a non-linearity to the output LMS signal to account for this, this is called activation function. Figure 40 shows how applying the *tanh* activation function results in the one-step ahead prediction task. It is clear the *tanh* function is not appropriate for this task by itself, the function varies between ± 1 and so fails to replicate the signal which varies between ± 40. $MSE = 196.7296$ shows how badly the prediction imitates the signal, and the prediction gain $R_p = -32.1899$ dB becomes negative due to the larger error in comparison to the predicted signal. If we observe Figure 40 we can observe how the prediction follows the signal accurately in term of trend, but fails to match the signal's amplitude due to the limitations of the *tanh* function. The *tanh* function by itself is not appropriate for the given task as the non-linearity fails to match the signal's amplitude.

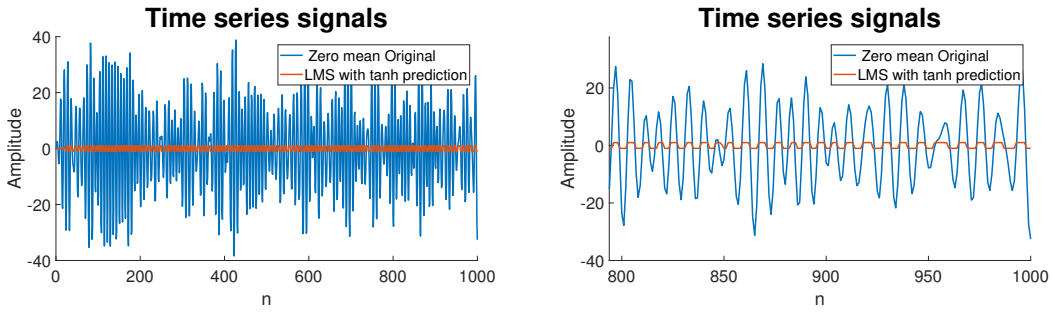


Figure 40: Original and LMS with *tanh* predicted time series signals. Left: complete, Right: zoomed

To deal with the mentioned issues of the *tanh* activation function, scaling can be applied in the form $a \cdot tanh$ to account for the signal's magnitude. This way the *tanh* function can apply the mentioned good trend following of the signal whilst the scaling can deal with the amplitude considerations. The zero-mean signal varies between ± 40 and so the value of a should be selected to account this. Suitable values of a would range between $35 : 45$ depending on how we wanna consider the higher amplitude values (Flatten or not by *tanh*'s saturation region), thus we pick $a = 40$ to analyse the effect of this in the one-step ahead prediction task. Figure 41 shows the effect of scaling by a of the *tanh* function, as can be seen the algorithm performs a very good task, and can be seen on the right plot the non linearity allows for the prediction to capture the signal better than the previous LMS (compare section around $n = 960$), providing the best prediction up to now. The substantially lower $MSE = 8.1420$ shows how the error drops significantly and the prediction gain $R_p = 14.0873$ dB reinforces the idea that the error has decreased and hence the ratio to the signal is more considerable. The LMS with scaled *tanh* performs much better than the original LMS capturing the non-linearity in the prediction task, the algorithm however has one issue, the need to decide the value of a. In order to set a the signal needs to be known beforehand to know its limits and so scale accordingly, this is not always the case and so this algorithm may not be suitable for all types of prediction tasks.

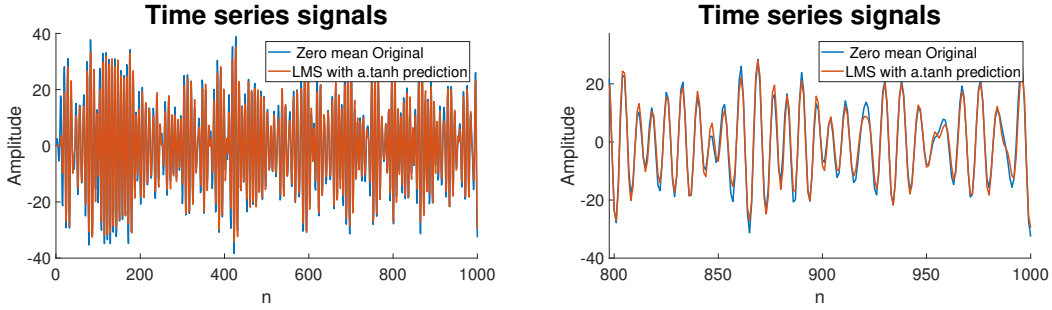


Figure 41: Original and LMS with $a \cdot tanh$ predicted time series. Left: complete, Right: zoomed

We now consider the prediction task on the original signal with non-zero mean. The previous discussed methods only work on zero mean data and so a bias needs to be introduced for correct prediction of the original signal, that is the system is $\phi(\mathbf{w}^{\mathbf{x}} + b$. In order to implement this we consider the augmented input to the algorithm $[1, \mathbf{x}]^T$, this way the weight vector also increases by one and adapts to adjust for the bias. By adding the additional 1 we are effectively adapting the algorithm so that $b = w_o \times 1$, and hence the predicted signal will account for the non-zero mean of the original signal. Figure 42 shows the implementation of this method, the performance of this method resembles very much the one from above which is exactly the same without the bias. The only difference is the initial struggle to achieve the correct weights due to the bias introducing an additional task. As can be seen the method fails to completely replicate the signal (higher amplitude samples), however does a very good job for the signal as a whole (right plot). $MSE = 13.7160$ and prediction gain $R_p = 12.2405$ dB, show how the algorithm performs slightly

worse than its zero-mean counterpart due to the extra difficulty to estimate the bias. The algorithm however achieves low MSE and high R_p proving it succeeds at the prediction task, only with a slower convergence than its zero-mean counterpart, as seen in Figure 42 (left) for the first samples.

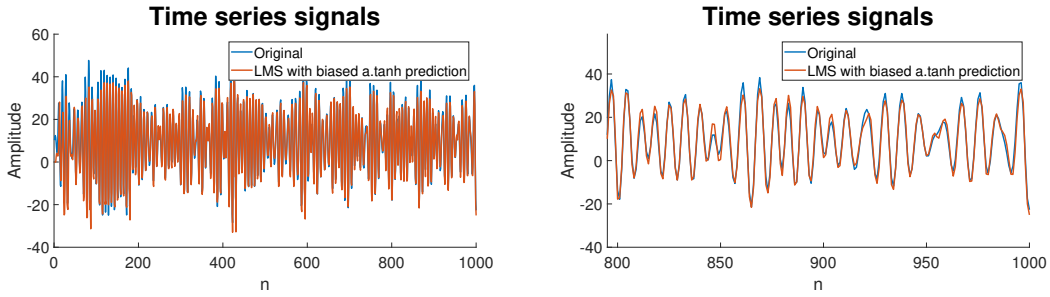


Figure 42: Original and biased LMS with $a \cdot tanh$ predicted series. Left: complete, Right: zoomed

A solution to the slow convergence time of the above methods is to pre-train the weights by overfitting a small smaple of the signal. As such we select the first 20 samples and with $w(0) = 0$ perform the LMS algorithm over 100 epochs. We thus obtain some initialisation of the weights according to the pre-training, which are then used in the LMS algorithm over the entire signal as weight initialisation. Figure 43 shows the results obtained when performing this method. The improvement is very clear in terms of convergence, the firm samples of the signal, unlike previously, are already properly estimated and the convergence of the method virtually zero. The method still performs successfully at the prediction task as before, see right plot, but now manages to converge much faster than previously, thus leading to the reduced $MSE = 6.9139$ and increased prediction gain $R_p = 15.1472$. This improved metrics reinforce the idea of an improved method over the biased non-linear LMS without pre-training.

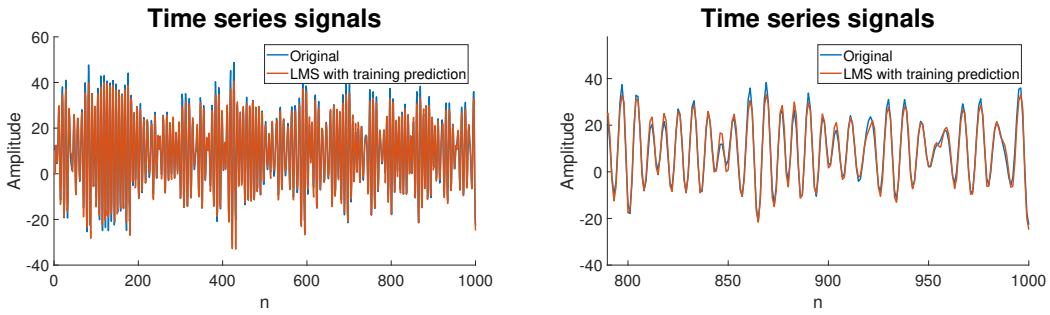


Figure 43: Original and non-linear LMS after training. Left: complete, Right: zoomed

A simple neuron however is not always capable of to express the highly non-linear nature of certain problems. As such various neurons (dynamical perceptrons) can be stacked together to form a deep network which manages to develop this complex mappings by setting the non-linear outputs of neurons as the input of other neurons, thus developing a higher non-linear mapping, an example is Figure 44.

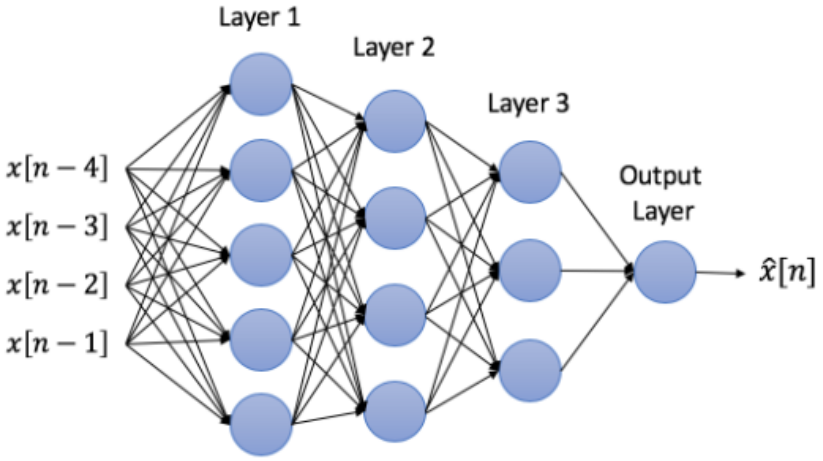


Figure 44: Example of a deep network with 3 hidden layers for predicting non-stationary time-series

Deep networks use *backpropagation* or backwards propagation to update the weights of the network according to the error at the output. This procedure involves propagating the error gradient backwards towards every weight in the network so that each gets updated accordingly, from this the need for differentiable activation functions is crucial to allow for the gradient to flow through the network. The method of backpropagation thus involves:

1. Feed Forward: For each neuron in the network compute $\mathbf{y} = \phi(\mathbf{w}^T\mathbf{x} + b)$
2. Compute the prediction error: $y - \hat{y}$
3. Backpropagate error: For each neuron on the network: compute the error on y, $\frac{\partial E}{\partial y_i}$, and then compute the error gradient on the weights, $\frac{\partial E}{\partial w_{ij}}$, to update the weights, $w_{ij}(n+1) = w_{ij}(n) - \eta\frac{\partial E}{\partial w_{ij}}$.

where $\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}\phi'(y_i)x_j$. Taking Figure 44 as an example, the network has $5+4+3+1 = 13$ neurons. Hence to update the weight leading from the 6^{th} neuron to the 10^{th} we need to compute all the previous gradients through which the error propagates.

$$\frac{\partial E}{\partial w_{6\to 10}} = \frac{\partial E}{\partial y_{13}}\frac{\partial y_{13}}{\partial w_{10\to 13}}\frac{\partial w_{10\to 13}}{\partial y_{10}}\frac{\partial y_{10}}{\partial w_{6\to 10}} \tag{50}$$

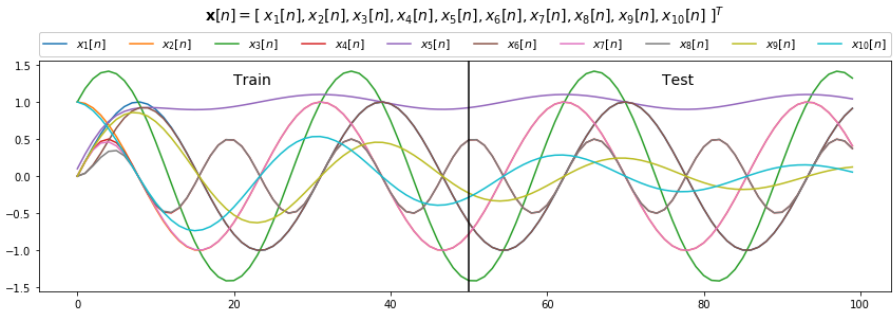


Figure 45: Components of signal x

The below examples show the task of predicting a signal corrupted by noise. The prediction task involves a highly non-linear mapping and so a deep network is developed for comparison with the simpler dynamical perceptron. The signal y is to be estimated using the input x as input,

Figure 45. Using 4 hidden layers, 20,000 epochs, learning rate of 0.01 and noise power equal to 0.05, we train the different predictors and evaluate their test performance in the task of signal prediction.

Figure 46 shows the comparison of the different methods in the prediction task. We can clearly see how the single neurons provide a smooth result, but fail to capture the signal's behaviour, whilst the deep network seems to predict the signal's trend, but doesn't properly predict it and exhibits flat regions which don't follow the original signal. From the loss plots, Figure 47, we can see how the different methods perform. The single neuron (linear and tanh) have a test error that decreases quickly and settles at around 0.1, signifying the convergence of the algorithms. The deep network test loss however varies significantly, showing how the network predicts successfully initially then increases error signifying overfitting, and then settles back down to around 0.9. From this we see how the use of neural networks may lead to better results (lower test loss), however oscillates significantly during training, as the network is too deep. All methods however show ovefitting, reaching the lowest test error at an early stage and then finally settling at a higher value after convergence.

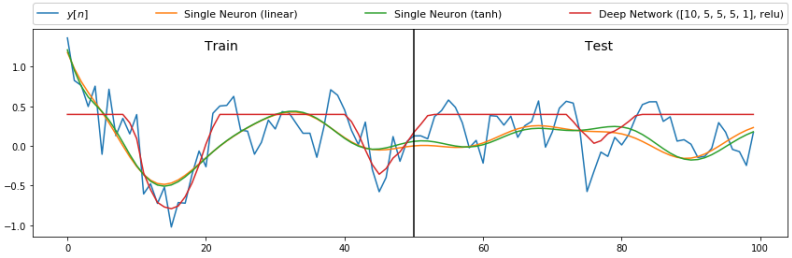


Figure 46: Signal prediction according to different methods

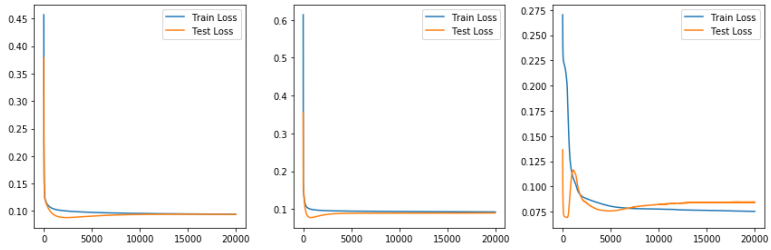


Figure 47: Loss plots for Linear single neuron (left), tanh single neuron (middle) and deep neural network (right)

We now vary the noise power to analyse the effect of a noisier signal on the prediction result. Figures 48 and 49 show the prediction and losses of the different methods for noise power equal to 0.5. From the prediction plot we can very clearly see how the noisy data makes the deep network strongly overfit the signal, without considering the presence of noise, this obviously has negative consequences on the test data were the smoother single neuron predictions are much more appropriate than the deep network prediction. From the loss plots we can again see the effect of larger noise power on the prediction algorithms, the more complex the method the more overfitting it suffers. The single neurons reach a minimum test error initially and then start to overfit leading to a higher test error than training error. The deep network shows again strong overfitting and oscillation in the last epochs. We can see how the complex deep network results in overfitting with the network trying to model the noise.

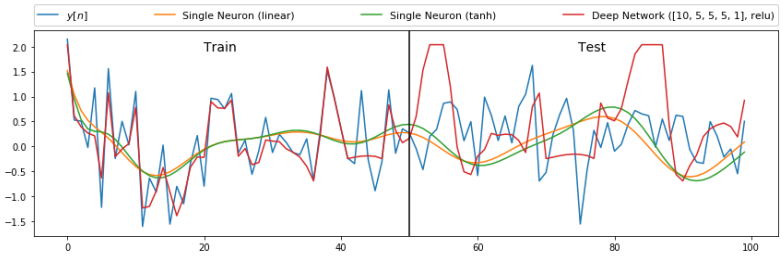


Figure 48: Signal prediction according to different methods

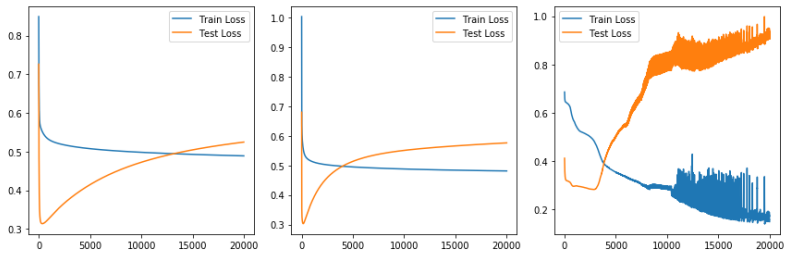


Figure 49: Loss plots for Linear single neuron (left), tanh single neuron (middle) and deep neural network (right)

Figures 50 and 51 show the prediction and losses of the different methods for noise power equal to 0.005. We can see that with a lower noise power the ability of the deep network is significantly better than before and manages to predict the test data very well. The loss plots show a similar story as for noise power set to 0.05, however with the lack of overfitting for single neurons. The deep network again oscillates up and down as before but manages to settle at a lower test loss than the single neurons (~ 0.5 vs ~ 0.9), showing its ability to properly predict the signal.

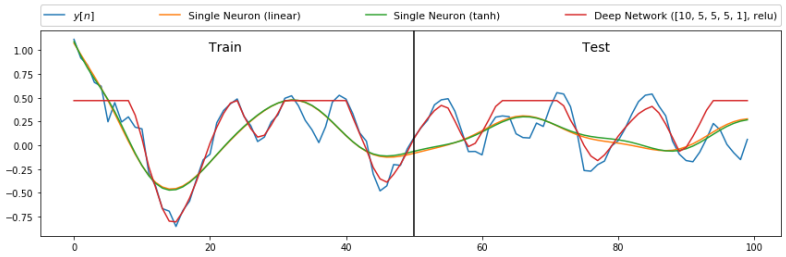


Figure 50: Signal prediction according to different methods

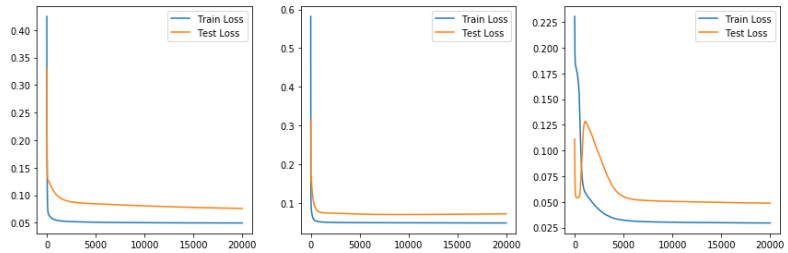


Figure 51: Loss plots for Linear single neuron (left), tanh single neuron (middle) and deep neural network (right)

The use of deep networks thus makes sense if the selected weights are that for the minimum loss, but leads to overfitting if too much training is done. Single neurons also exhibit this, however are much smoother showing the lower complexity of these methods. From the plots we can see how deep networks tend to overfit and so huge noise power affects them most, as the network attempts to model the noise. The inability to generalise by deep networks make them a bad choice for prediction of noisy data, however their higher complexity make them a great choice to model complex noiseless signals. The simpler single neurons provide a more simpler and smoother prediction for the signal under the different levels of noise, thus the conclusion is inverse. Under a large noise power these methods generalise well and provide a good estimate, however under little noise they fail to properly capture the signal's behaviour and so are outperformed by more complex methods.