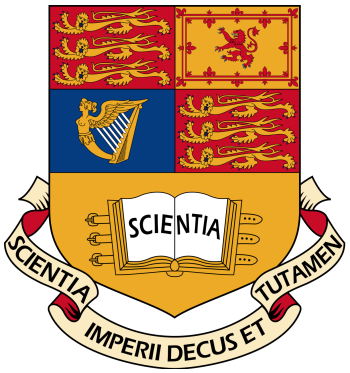


Imperial College London

Department of Electrical and Electronic Engineering

Adaptive Signal Processing and Machine Intelligence Coursework

Adaptive Signal Processing and Machine Intelligence

ELEC97002 - 2021-2022

Author Name: Manginas Vasileios
Author Email: vm3218@ic.ac.uk
Author CID: 01542774

Contents

1 Classical and Modern Spectrum Estimation

1.1 Properties of Power Spectral Density (PSD)

The series of equations below explores the equivalence of direct and indirect methods of power spectral density (PSD) estimation. We start from the formula of the direct method (Definition 2).

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

$$= \lim_{N\to\infty} \mathbb{E}\left\{ \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{-jn\omega} \sum_{m=0}^{N-1} x^*(m) e^{jm\omega} \right\} \tag{1.2}$$

$$= \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} \mathbb{E}\left\{ x(n) x^*(m) \right\} e^{-j(n-m)\omega} \tag{1.3}$$

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

$$= \lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} (N-|k|) r(k) e^{-j\omega k} \tag{1.5}$$

$$= \lim_{N\to\infty} \left(\sum_{k=-(N-1)}^{N-1} r(k) e^{-j\omega k} - \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k| r(k) e^{-j\omega k} \right) \tag{1.6}$$

$$= \sum_{k=-\infty}^{\infty} r(k) e^{-j\omega k} - \lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k| r(k) e^{-j\omega k} \tag{1.7}$$

By introducing the assumption that the covariance $r(k)$ decays rapidly as shown in Equation 1.8, we obtain the indirect method for PSD estimation (Definition 1) in Equation 1.9.

$$\lim_{N\to\infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k||r(k)| = 0 \tag{1.8}$$

$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k) e^{-j\omega k} \tag{1.9}$$

What we refer to as the "indirect method" is in fact the discrete-time case of the Wiener–Khinchin theorem, which claims that under the assumption that equation 1.8 holds, and thus that the autocovariance (ACF) $r(k)$ is absolutely integrable, the ACF and the PSD of a signal are a pair related by the DTFT.

In the simulations depicted in Figure 1 we estimate the PSD of an impulse and of a sum of two sinusoidals by using both direct and indirect methods. In the case of the impulse, the autocovariance function is also an impulse. Equation 1.8 is clearly satisfied and the equivalence between direct and indirect methods holds. However, for the sum of the sinusoidals the autocovariance does not decay as rapidly. This introduces a non-zero error term in Equation 1.7, subsequently resulting in a discrepancy between the PSD spectra computed from the direct and indirect methods.

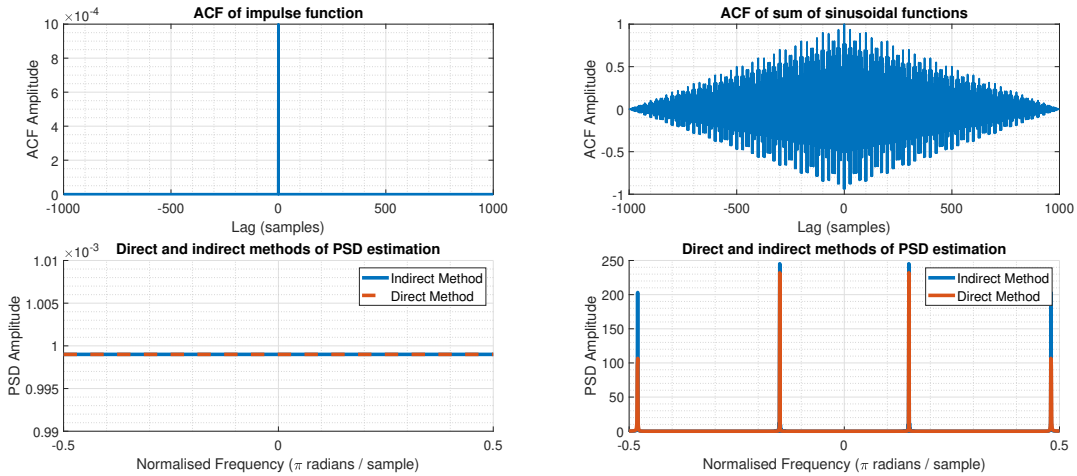


Figure 1: Autocovariance function (top) and estimation of PSD using direct and indirect approaches (bottom) for impulse function (left) and a sum of sinusoidal functions (right)

1.2 Periodogram-based Methods Applied to Real–World Data

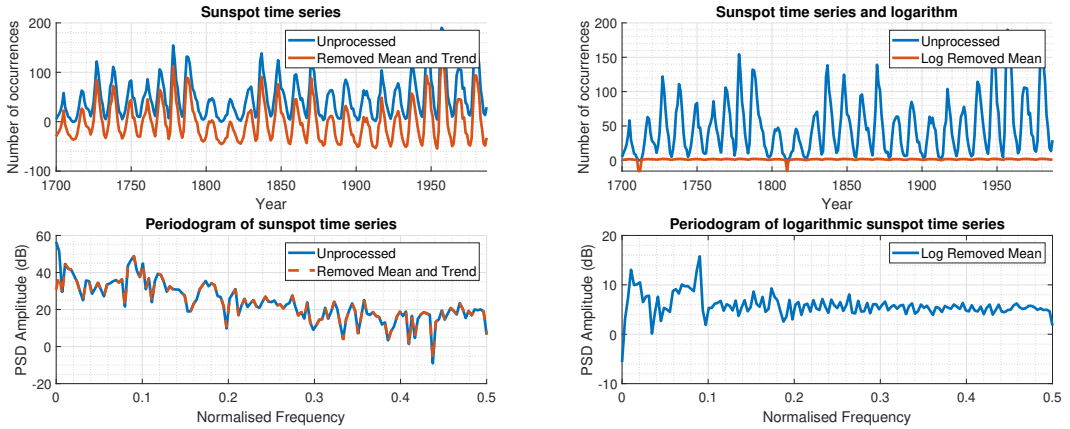


Figure 2: Time series (top) and periodogram (bottom) of unprocessed and processed sunspot data

In Figure 2 we have applied the periodogram to the sunspot time series. We observe that removing the mean and trend from the time domain data heavily reduces the low frequency components in the corresponding spectrum. This is expected as *mean* centres the signal around 0, thus removing the DC component of the spectrum, while *detrend* subtracts the line of best fit. When applying the logarithm to the data and removing the corresponding mean, we observe that the signal spikes when the original signal experiences less periodicity, resulting in much more defined peaks in the periodogram of the logarithmic data.

In Figure 3 we compare the performance of the standard periodogram against the averaged periodogram, as applied to the POz EEG signal, and with the aim of determining the SSVEP frequency. While we observe that both approaches manage to exhibit the SSVEP occuring at 13Hz, the averaged periodogram has much more well-defined peaks at the corresponding frequencies, and performs much better in rejecting the underlying EEG activity of the surrounding spectrum. The tiredness-induced alpha-rhythm (8-10Hz), harmonics of the SSVEP (26, 39Hz), as well as the mains interference (50Hz), are all more pronounced and distinguishable in the averaged case. That said, as the window length is decreased from 10s to 1s in the averaged periodogram, the spectrum is smoothened out, drastically reducing the sharpness of the peaks. We could even say that the averaged periodogram with the smallest window is a less insightful representation than the standard periodogram.

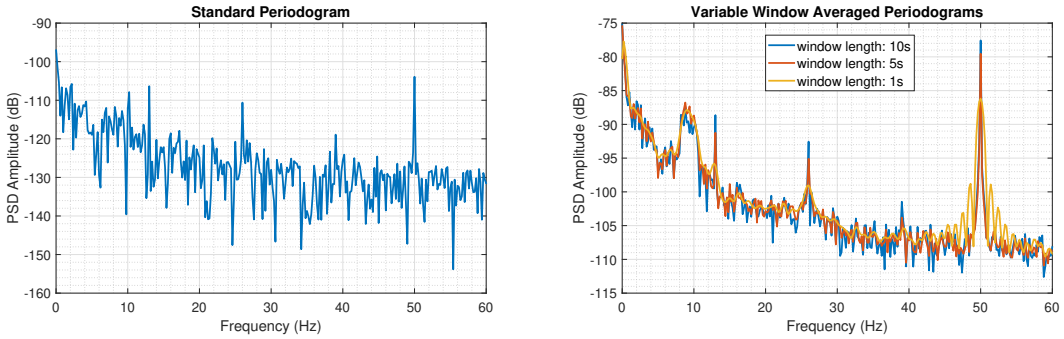


Figure 3: Standard periodogram (left) and averaged periodogram with different window lengths (right) for EEG data

1.3 Correlation Estimation

In Section 1.1 we used a biased autocorrelation estimator. Equation 1.3 shows both the biased, and unbiased versions of the autocorrelation estimators.

$$r(k) = \begin{cases} \frac{1}{N} \sum_{n=k+1}^{N} x(n)x^*(n-k), & \text{biased} \\ \frac{1}{N-k} \sum_{n=k+1}^{N} x(n)x^*(n-k), & \text{unbiased} \end{cases} \tag{1.10}$$

In Figure 4, we investigate the differences between these two approaches in producing the correlogram and the corresponding spectrum of three signals, a white gaussian noise (WGN) signal, a filtered WGN signal, and a sum of two noisy sinusoidal signals. As expected, we observe that while the correlograms of the two estimators are almost identical in the beginning (up to around sample 20), the unbiased ACF estimate behaves more and more erratically as the lag moves closer to the total number of samples. As fewer samples are available for estimation, the ACF of the unbiased estimator is not positive definite, leading to negative values for the PSD, as can be seen in the rightmost plots of the figure. In general, the biased version appears to produce much smoother spectra, and is therefore proposed to be more robust.

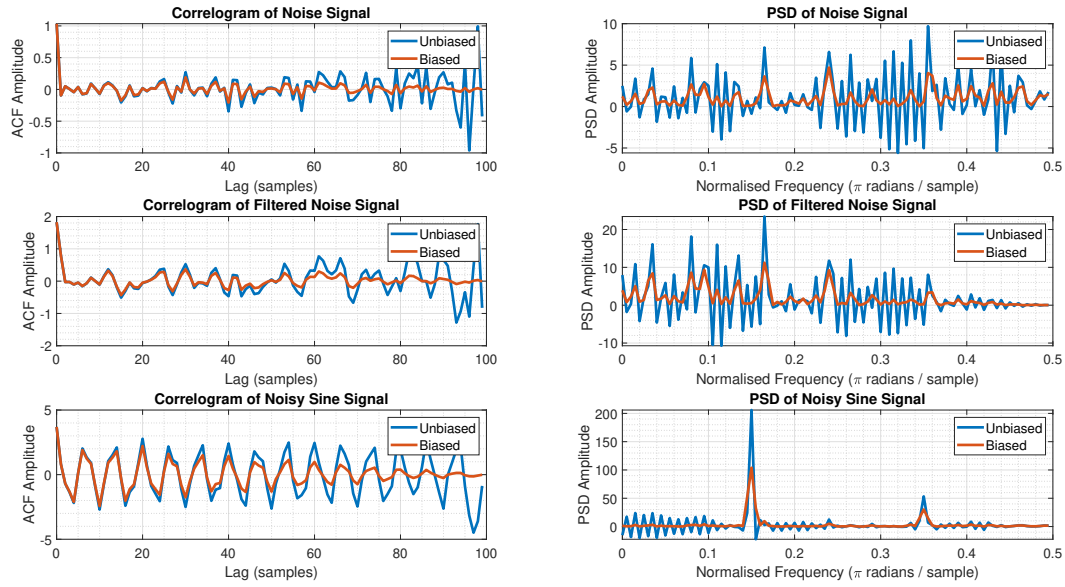


Figure 4: Unbiased and biased correlograms (left) and corresponding PSD spectra (right) of pure noise (top), filtered noise (middle), and noisy sine signal (bottom)

We now use the biased estimator to generate a PSD estimate of 100 realisations of a noisy process. In this case, this is a sum of two noisy sinusoids with different amplitudes and frequencies at 0.1Hz and 0.35Hz. From the left side of Figure 5 we can see that although the individual realisations produce variations throughout the spectrum, the mean of the realisations very closely follows the PSD of the clean signal. The plot in the bottom indicates that standard deviation is proportional to PSD amplitude.

The right side of the figure exhibits the same plots but with dB scaling. We see that even though the sine peaks are still clear, the variations of the individual realisations are much greater in magnitude. This is expected since the logarithm exaggerates small fluctuations at values close to zero. However, the standard deviation in this case has minima rather than a maxima at PSD peaks. This can also visually be seen from the PSD plot, since the individual realisations are much less spread out at PSD peaks. Therefore, we can say that the benefit of the dB representation, contrary to the non-dB version, is that it doesn't suffer from the fact that standard deviation of individual realisations is higher at the frequencies of interest, namely the PSD peaks.

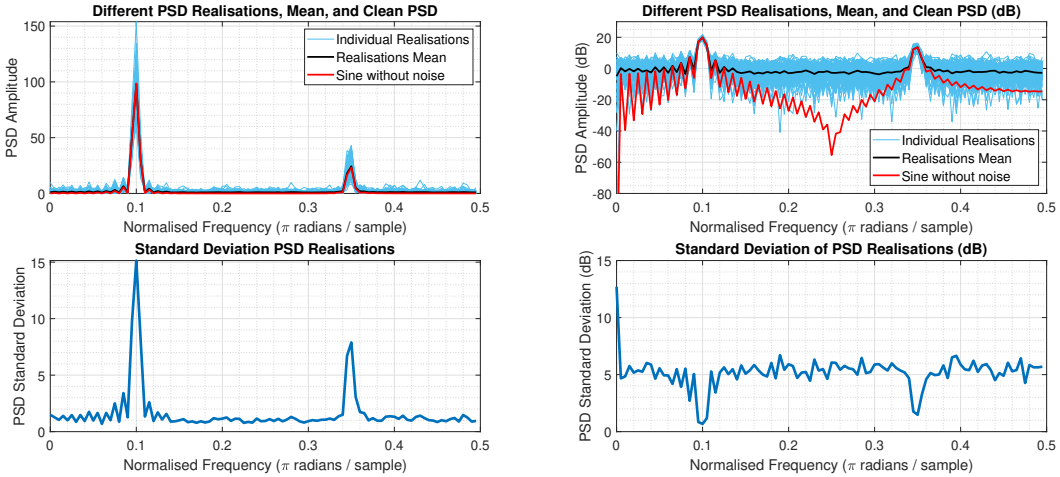


Figure 5: PSD spectra of sum of two sinusoidals: 100 individual realisations, realisations mean, clean signal PSD, and standard deviation of PSD realisations. Shown in non-dB (left) and dB (right) scale

In Figure 6 the signal we use consists of the sum of two complex exponentials closely spaced in frequency plus noise. We investigate the effect of signal lengths on the periodogram estimation. While the PSD is correct for a signal length of 100 samples, for signal lengths of less than 50 samples the periodogram does not exhibit the correct peaks. This makes sense, as the resolution of the periodogram is proportionate to 1/N.

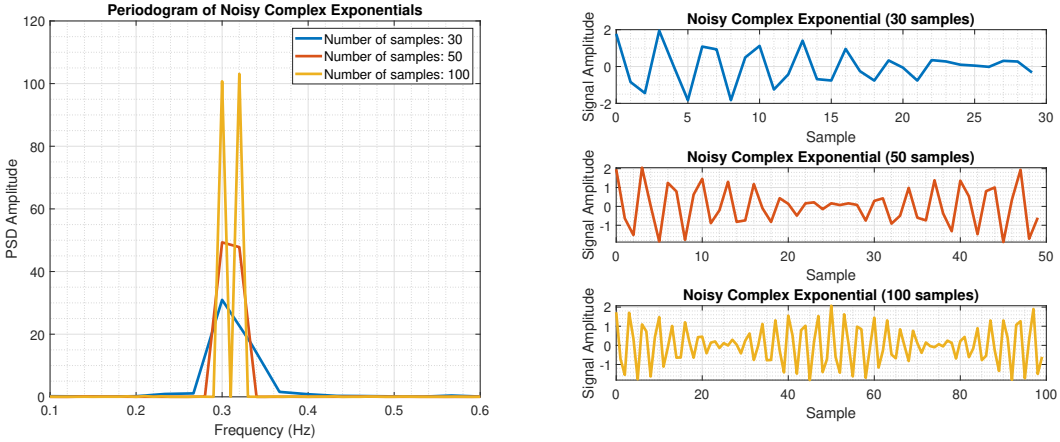


Figure 6: Sum of two complex exponentials with closely-spaced frequencies (right) and corresponding periodogram (left) for different number of samples

Next, we investigate the performance of the MUltiple SIgnal Classification (MUSIC) algorithm. In code, this is achieved with the following block:

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

The first line is used to compute a biased estimate of the autocorrelation matrix. Regarding the *corrmtx* function arguments, x is the input, in this case the sum of the noisy exponentials, and 14 is the prediction model order which specifies that the autocorrelation matrix estimate $\mathbf{R}$ is going to be $15x15$. Finally, 'mod' refers to the method used, the modified covariance method, which uses forward and backward prediction error estimates, based on the 14^{th}-order prediction model. The output $\mathbf{X}$ is a rectangular Toeplitz matrix, while $\mathbf{R}$ is the autocorrelation matrix, computed as $\mathbf{X}^H\mathbf{X}$.
The second line implements the MUSIC algorithm to produce the pseudospectrum estimate, $\mathbf{S}$, as evaluated at the normalised frequencies specified in the vector $\mathbf{F}$. The final argument, 'corr', specifies that the input argument, $\mathbf{R}$ in this case, is to be interpreted as a correlation matrix rather than matrix of signal data. 2 is the signal space dimensionality, [ ] uses the default number of DFT points, and 1 specifies the sampling frequency. Finally, lines 3-7 create the MUSIC pseudospectrum plot, limiting the x-axis in the $[0.25, 0.40]$ frequency range.

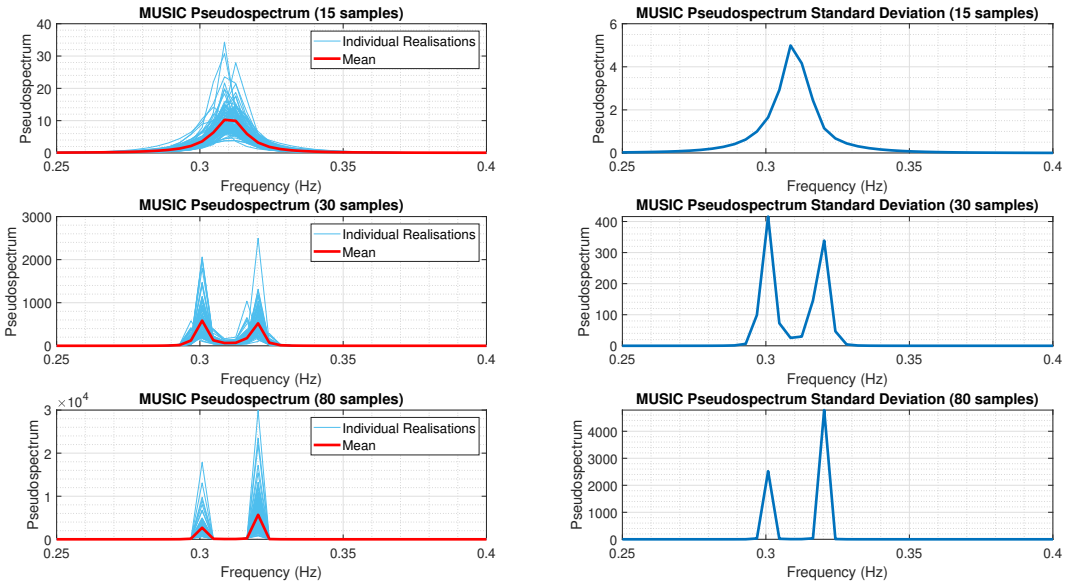


Figure 7: Pseudospectrum estimation with MUSIC: 100 individual realisations and realisations mean (left) and standard deviation of realisations (right) for different number of samples

Figure 7 depicts the performance of MUSIC in estimating the periodogram of the noisy, closely-spaced exponentials, showing individual realisations, their mean, as well as the standard deviation. We observe that performance is better than the periodogram and that MUSIC provides more detailed information, since it is able to discern the correct peaks even at 30 samples, whereas the standard periodogram was not. Once again, we see that peaks in standard deviation occur together with peaks of the PSD. The main disadvantage of MUSIC lies in the fact that it requires knowledge of the subspace dimensionality, meaning that it is much less effective for making general spectrum estimates.

1.4 Spectrum of Autoregressive Processes

Finding the AR parameters requires that the ACF matrix used in the set of Yule-Walker equations is invertible. Since autocorrelation matrices are by construction symmetric, the condition for such matrices to be invertible is that they are positive definite. As mentioned in Section 1.3 the unbiased estimator does not guarantee that the ACF matrix is positive definite, meaning that it is an unwise option for finding the AR parameters.

In Figure 8 we depict the PSD estimates for different orders of the assumed underlying model of the signal. The true signal comes from an AR(4) process as specified from the equation below.

$$x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n) \tag{1.11}$$

The left side of the figure investigates a signal of length 1000 samples (in reality 500 samples as the first 500 samples are neglected to remove the transient response of the filter), and shows the PSD estimation for the best-performing model orders, the true PSD, as well as a plot of MSE against model order. We observe that the error reduces as the order increases, and that low-order models are unable to capture the PSD peaks correctly. After the order at which the MSE is minimised, a plateau is reached. The right side of the figure uses the same signal but with 10000 (in reality 9500) samples. We see that the error is lower throughout the range of model orders, and that the minimum is reached at a smaller order. That said, when the chosen model is lower than the correct (AR(4)) order, the PSD is not described accurately. After this, though, the error again plateaus.

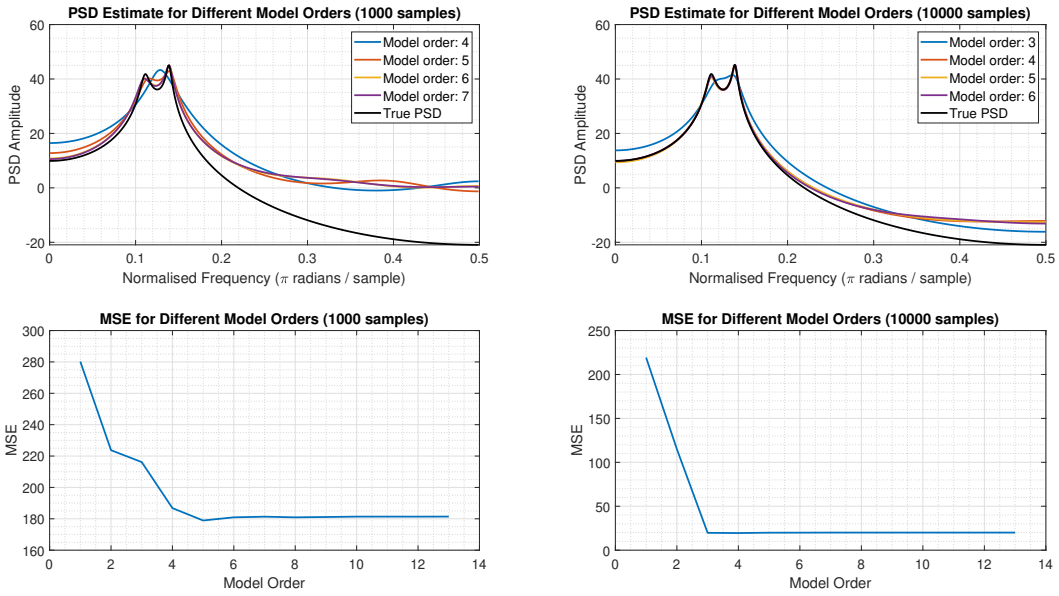


Figure 8: PSD estimate for different AR model orders (top) and corresponding estimation MSE (bottom) for data lengths of 1000 (left) and 10000 (right)

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

In this section we apply concepts from the previous sections to real-world data ECG data with the aim of extracting the rate of respiration. In Figure 9 below we plot the periodograms for the three different trials of RRI data. Our aim is to observe peaks in the power spectrum indicating a particular breathing frequency b_f. The corresponding breathing rate in breaths per minute (BPM) and the is given by $BPM = 2 * 60 * b_f$.

	RRI trial 1	**RRI trial 2**	**RRI trial 3**
$\mathbf{b_f}$	-	0.416	0.125
BPM	Unconstrained	50	15

The results are summarised in the table above. Note that in the first trial the subject of the experiment was not breathing with a fixed rate and therefore the PSD of the RRI data does not show a definitive peak. From the remaining entries of the table we can conclude that the second trial involved fast breathing, while the third exhibits normal breathing rate.

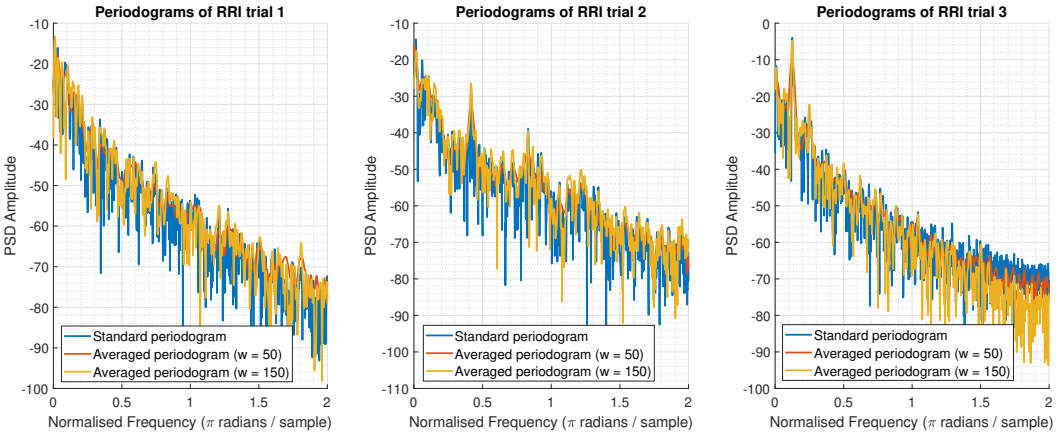


Figure 9: Periodograms for the three RRI data trials

Next, we aim to generate AR-based power spectra by fitting AR processes of different orders to the RRI data of each trial. Figure 10 depicts the same standard periodograms from before along with the spectrum generated by the AR process. For each of the trials the AR model order used is different, and was found by plotting several orders and picking the one with least complexity that first exhibits a peak at the expected breathing frequency for that trial.

In the first trial where there is no definitive peak an AR(5) model is able to model the PSD relatively well, although this is not significant. In trials 2 and 3 AR(20) and AR(25) models were used respectively. In both cases, the models where able to detect the PSD peaks accurately. Additionally, this process also denoises the PSD to an extent. However, this comes at the price of the frequency resolution, as well as potential inaccuracies, as seen in the higher frequencies of trial 3.

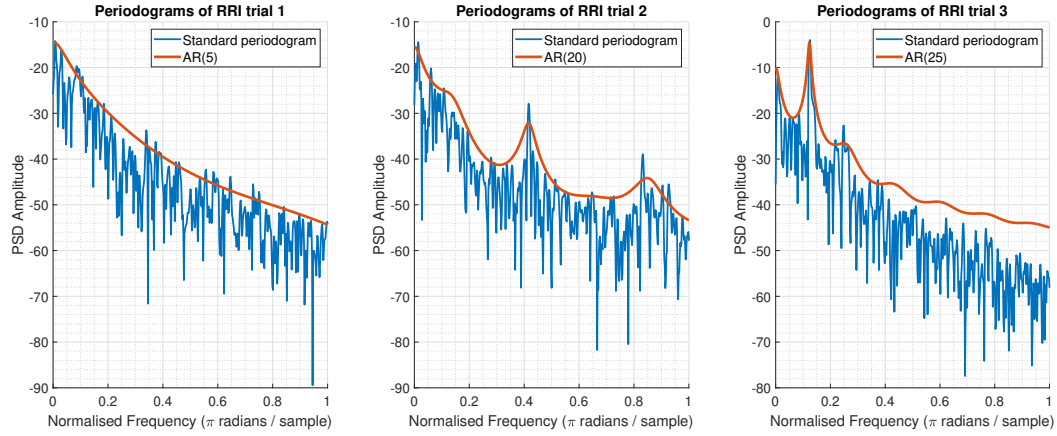


Figure 10: Periodogram and AR spectrum estimation for the three RRI data trials

1.6 Robust Regression

Figure 11 depicts the singular values of the input matrix $\mathbf{X}$ and of the noisy input matrix $\mathbf{X_{noise}}$, and the squared error between them. We can clearly conclude that the rank of the input data is equal to 3, since there are 3 singular values, while the noisy input matrix is full rank. By looking at the squared error between them,

we understand that the effect of the added noise on the singular values of the input is a disturbance from their nominal values. Under the current configuration, this disturbance is small, making it easy to identify its rank. However, we can speculate that if the noise was larger in magnitude (i.e. the Signal-to-Noise Ratio (SNR) was lower), then this disturbance might be large enough to make it difficult to identify the input rank.

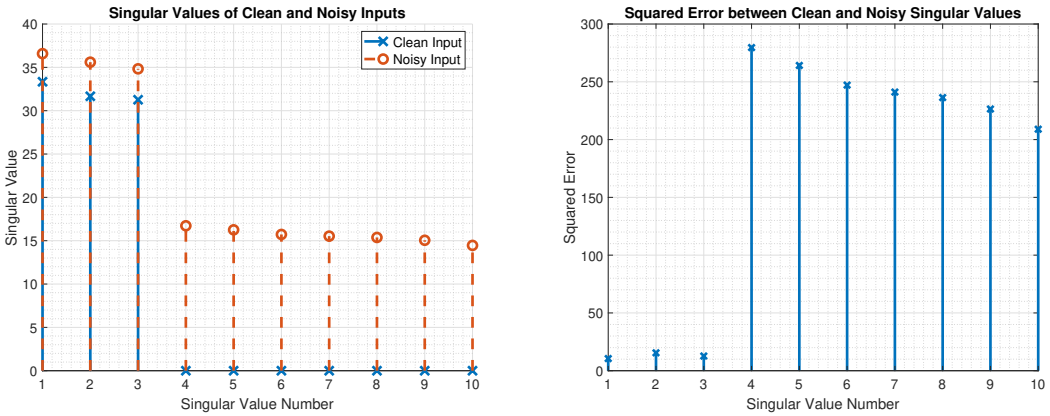


Figure 11: Singular values of clean and noisy input data (left) and squared error between them (right)

In Figure 12 we plot the difference between the columns of the noiseless input and of the noisy input, as well between the noiseless input and a denoised version of the noisy input, generated by reconstructing the matrix using only the first 3 principal components. We can clearly observe that the error in the denoised case is much smaller, and that therefore it is a better approximation of the input than the noisy version. That said, the standard deviation of the error in the denoised case is also larger.

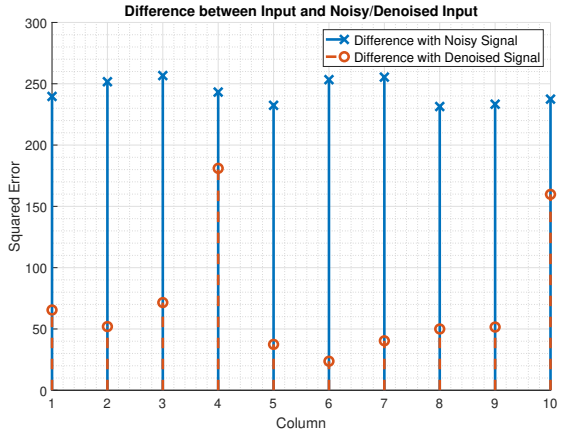


Figure 12: Difference between columns of clean input and columns of noisy input and "denoised" (low-rank approximation) noisy input

The output matrix $\mathbf{Y}$ is given by the expression $\mathbf{Y} = \mathbf{XB} + \mathbf{N_Y}$, where the matrix $\mathbf{B}$ is unknown. Both Ordinary Least Squares (OLS) and Principal Component Regression (PCR) are methods aiming at approximating this matrix. The different approaches are specified in the equations below. In the PCR case the matrices $\mathbf{V}_{1:r}, \mathbf{\Sigma}_{1:r}, \mathbf{U}_{1:r}$ are low-rank approximations of the matrices from the SVD of the noisy input.

$$\hat{\mathbf{B}}_{\text{OLS}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y} \tag{1.12}$$

$$\hat{\mathbf{B}}_{\text{PCR}} = \mathbf{V}_{1:r}\left(\mathbf{\Sigma}_{1:r}\right)^{-1}\mathbf{U}_{1:r}^T\mathbf{Y} \tag{1.13}$$

Figure 13 depicts the performance of the two algorithms in terms of square error in the training as well as test set. The training set is used to create the estimates for the unknown matrix **B**, while the test set is used to evaluate the performance on unseen data.

Method	Training Error	Test Error
OLS	3552	2377
PCR	3577	2352

We observe that both algorithms perform very similarly, exhibiting almost identical errors in both training and test sets. That said, from the table above we can see the minor differences in the summed error along all columns; PCR performs slightly better in the training set, and OLS performs slightly better in the test set. That said, both these differences appear negligible.

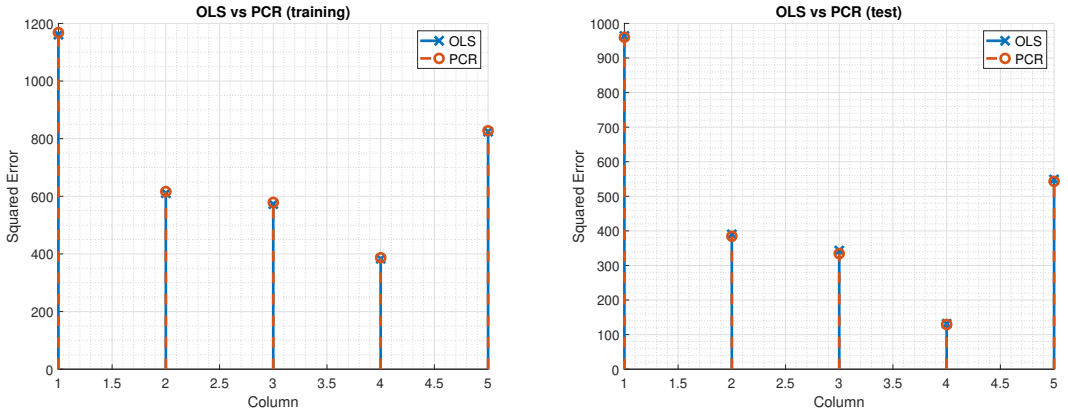


Figure 13: Estimation error of OLS vs PCR schemes

Naturally, whenever there is noise existent, testing largely benefits from collecting metrics over an ensemble of data rather than from a single realisation. Figure 14 shows the average performance of these algorithms across 1000 trials. Surprisingly, the two methods appear to consistently exhibit almost identical results over a larger ensemble of data. Again, there is a very small advantage, with overall MSE of OLS being 2358, while that of PCR was 2337. However, as before, these differences appear insignificant.

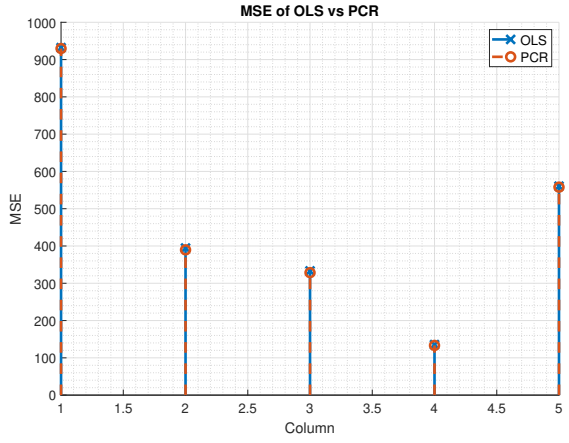


Figure 14: Estimate error of OLS vs PCR schemes in ensemble test with 1000 trials

2 Adaptive Signal Processing

2.1 The Least Mean Square (LMS) Algorithm

The correlation matrix of the input vector $\mathbf{x}(n) = [x(n-1), x(n-2)]^T$ is given by the expression below. Here r_{xx} refers to the ACF of the input $\mathbf{x}(n)$. As the AR process generating the input is stationary on its own, the aucorrelation can be expressed solely in terms of the time lag k.

$$\mathbf{R}_{xx} = \mathbb{E}\left\{\mathbf{x}(n)\mathbf{x}(n)^T\right\} \tag{2.1}$$

$$= \mathbb{E}\left\{ \begin{array}{cc} x(n-1)x(n-1) & x(n-1)x(n-2) \\ x(n-2)x(n-1) & x(n-2)x(n-2) \end{array} \right\} \tag{2.2}$$

$$= \left(\begin{array}{cc} r_{xx}(0) & r_{xx}(1) \\ r_{xx}(-1) & r_{xx}(0) \end{array} \right) \tag{2.3}$$

Solving for the ACF:

$$r_{xx}(k) = \mathbb{E}\{x(n)x(n-k)\} \tag{2.4}$$

$$= \mathbb{E}\left\{[a_1 x(n-1) + a_2 x(n-2) + \eta_n]\, x(n-k)\right\} \tag{2.5}$$

$$= a_1\mathbb{E}\{x(n-1)x(n-k)\} + a_2\mathbb{E}\{x(n-2)x(n-k)\} + \mathbb{E}\{\eta(n)x(n-k)\} \tag{2.6}$$

$$\Rightarrow r_{xx}(k) = \begin{cases} a_1 r_{xx}(1) + a_2 r_{xx}(2) + \sigma_\eta^2, & k = 0 \\ a_1 r_{xx}(k-1) + a_2 r_{xx}(k-2), & k \neq 0 \end{cases} \tag{2.7}$$

Therefore, for $k = 0, 1, 2$ we obtain the following system of equations:

$$r_{xx}(0) = a_1 r_{xx}(1) + a_2 r_{xx}(2) + \sigma_\eta^2 \tag{2.8}$$

$$r_{xx}(1) = a_1 r_{xx}(0) + a_2 r_{xx}(-1) \tag{2.9}$$

$$r_{xx}(2) = a_1 r_{xx}(1) + a_2 r_{xx}(0) \tag{2.10}$$

Since the ACF is symmetric by definition, $r_{xx}(1) = r_{xx}(-1)$, and because the values of a_1, a_2, σ_η^2 are known, the system has three equations and three unknowns. Therefore, using $a_1 = 0.1, a_2 = 0.8, \sigma_\eta^2 = 0.25$ we obtain:

$$r_{xx}(0) = \frac{25}{27} \tag{2.11}$$

$$r_{xx}(1) = \frac{25}{54} \quad \Rightarrow \quad \mathbf{R}_{xx} = \frac{25}{54}\left(\begin{array}{cc} 2 & 1 \\ 1 & 2 \end{array} \right) \tag{2.12}$$

$$r_{xx}(2) = \frac{85}{108} \tag{2.13}$$

The condition on the step size μ for convergence is

$$0 < \mu < \frac{2}{\lambda_{\max}} \tag{2.14}$$

Here $\lambda_{\max}$ refers to the largest eigenvalue of the autocorrelation matrix. Eigendecomposition of $\mathbf{R}_{xx}$ gives $\lambda_1 = 0.463, \lambda_2 = 1.389$. Therefore, the range of μ for convergence is:

$$0 < \mu < \frac{2}{1.389} \quad \Rightarrow \quad 0 < \mu < 1.44 \tag{2.15}$$

In Figure 15 we depict the learning curve of the LMS algorithm for different step sizes. The learning curve here refers to the average of the squared error curves in dB scaling over an ensemble of input data, in this case 100 trials. Evidently, the LMS with step size $\mu = 0.05$ converges faster than the one with $\mu = 0.01$. This is expected, since a larger step in gradient-based algorithms intuitively leads faster to the minimum. Naturally, this argument does not hold universally since it dismisses realistic issues such as local minima, oscillatory behaviour, and divergence induced by large step size just to name a few.

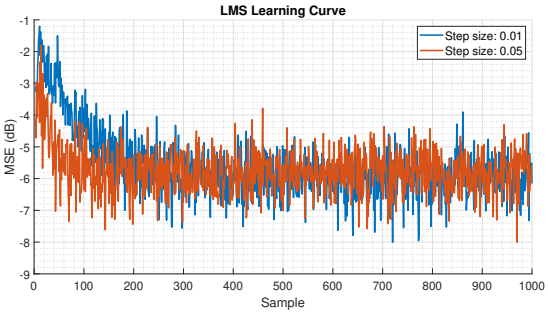


Figure 15: Learning curve of LMS predictor for estimation of an AR(2) process using different step sizes

Excess Mean Square Error (EMSE) is a measure used to quantify the difference between the mean-square error introduced by adaptive filters and the minimum attainable mean square error of a Wiener filter. The corresponding misadjustment, $\mathcal{M}$, is computed as the ratio between the excess mean square error and the minimum mean square error and is thus given by $\mathcal{M} = \frac{\text{EMSE}}{\sigma_\eta^2}$. In Table 1 we provide values for the EMSE and corresponding misalignment for different step sizes and over 100 individual trials. Additionally, for small step sizes a theoretical approximation of the misalignment exists, and is also given in the table. As expected, we observe that the approximation is more accurate in the case of the smaller step size.

Step Size ($\boldsymbol{\mu}$)	**EMSE**	**Misalignment** ($\mathcal{M}$)	**Approximated Misalignment** ($\mathcal{M}_{LMS}$)
0.01	0.0024	0.0096	0.0093
0.05	0.0131	0.0525	0.0463

Table 1: Experimental EMSE and misalignment estimates as well as theoretical misalignment approximation for different step sizes

In Figure 16 we plot the progression of the adaptive filter coefficients for different step sizes and averaged over 100 independent trials of the experiment. Note that the correct coefficients are $a_1 = 0.1, a_2 = 0.8$. We observe that the higher step size $\mu = 0.05$ leads to faster convergence as aforementioned, but also results in a higher steady state error than the smaller step size $\mu = 0.01$. While for the coefficient a_1 both step sizes appear to settle accurately on 0.1, for a_2 the smaller step size settles around 0.77, whereas the larger step size settles around 0.7. Furthermore, the steady state error variance is evidently larger for the larger step size. This introduces a tradeoff between good convergence speed and acceptable steady state error variance.

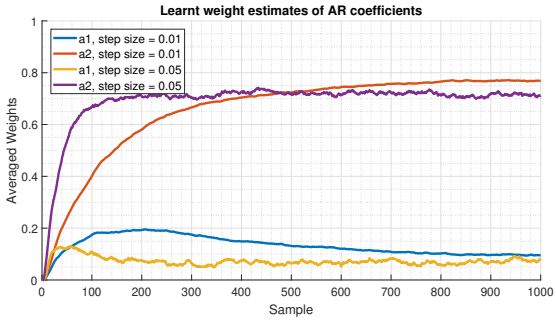


Figure 16: Evolution of averaged LMS weights for an AR(2) process using different step sizes

In the case where the autocorrelation matrix of the input has zero eigenvalues, there are weights of the LMS algorithm which do not converge. One way to tackle this is to introduce a leakage coefficient, γ, which forces the weights to converge. To obtain the weight update equation for the leaky LMS algorithm we begin by minimising the following cost function:

$$J_2(n) = \frac{1}{2}\left(e^2(n) + \gamma\|\mathbf{w}(n)\|_2^2\right) \tag{2.16}$$

$$= \frac{1}{2}\left[\left(x(n) - \mathbf{w}(n)^T\mathbf{x}(n)\right)^T\left(x(n) - \mathbf{w}(n)^T\mathbf{x}(n)\right) + \gamma\mathbf{w}(n)^T\mathbf{w}(n)\right] \tag{2.17}$$

Take the derivative with respect to the weight vector:

$$\frac{d[J(n)]}{d[\mathbf{w}(n)]} = -\left(x(n) - \mathbf{w}(n)^T\mathbf{x}(n)\right)\mathbf{x}(n) + \gamma\mathbf{w}(n) \tag{2.18}$$

$$= -e(n)\mathbf{x}(n) + \gamma\mathbf{w}(n) \tag{2.19}$$

Substituting within the generic weight update equation we obtain the expected result.

$$\mathbf{w}(n+1) = \mathbf{w}(m) + \mu\left(-\frac{d[J(n)]}{d[\mathbf{w}(n)]}\right) \tag{2.20}$$

$$= \mathbf{w}(n) + \mu(e(n)\mathbf{x}(n) - \gamma\mathbf{w}(n)) \tag{2.21}$$

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

In Figure 17 we investigate the performance of this algorithm in estimating the correct coefficients for different values of the leakage parameter, γ. In this case we see that the leakage coefficient has a negative effect overall on the model performance, since the algorithm performs worse than the non-leaky version for all values. Additionally, as γ increases the adaptive filter coefficients settle further and further away from their correct values. This can be explained through the fact that the leaky LMS algorithm is constructed in order to ensure invertibility of the matrix in the Wiener weights solution equation: $\mathbf{w} = \mathbf{R}^{-1}\mathbf{x}$. While normally $\mathbf{R}$ is not guaranteed to be invertible, the leaky LMS substitutes this for $\mathbf{R} - \gamma\mathbf{I}$, which is invertible, ensuring the existence of the solution. However, in ensuring this fact, the algorithm introduces a deviation proportional to the leakage coefficient. Thus, as γ increases, so does the discrepancy.

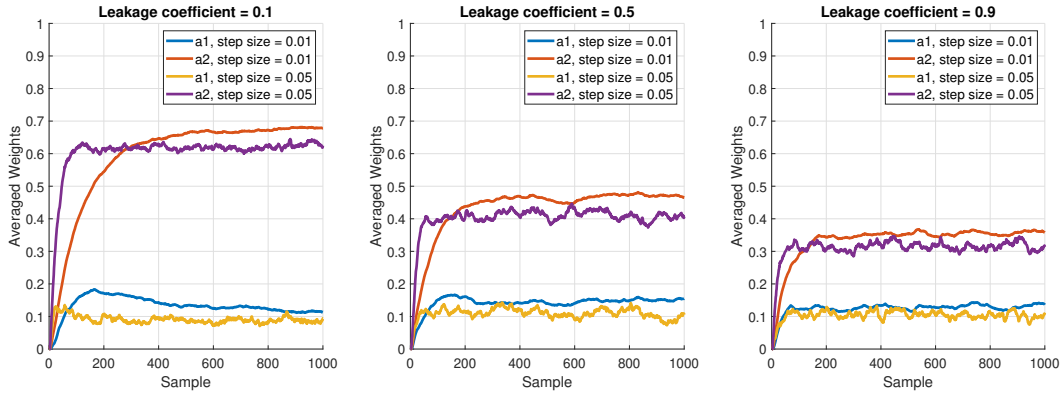


Figure 17: Evolution of averaged Leaky LMS weights for an AR(2) process using different step sizes and leakage coefficients

2.2 Adaptive Step Sizes

As mentioned in the previous section, the choice of a fixed step size comes with a tradeoff of convergence speed against steady state error variance. This is the issue variable step-size (VSS) algorithms aim to solve. In particular, gradient adaptive step-size (GASS) algorithms vary the step size adaptively according to the update equation below, where $\psi(n)$ differs from algorithm to algorithm.

$$\mu(n+1) = \mu(n) + \rho e(n)\mathbf{x}^T(n)\psi(n) \tag{2.23}$$

In this case, we experiment with three such algorithms: Benveniste, Ang & Farhang, and Matthews & Xie. Figures 18 and 19 depict the weight error for these algorithms, as well as for two standard LMS schemes with different step sizes. The difference between the two figures is the value of the parameter ρ from Equation 2.2, which is set to 0.001 for Figure 18 and 0.005 for Figure 19. Finally, note that the desired signal is a MA(1) process and that the initial step sizes for all GASS algorithms was set to 0.

In the first figure, where $\rho = 0.001$, we observe that the LMS algorithm with the higher step size converges fastest out of all algorithms, while the LMS algorithm with the lower step size has the slowest convergence. For the second figure, where $\rho = 0.005$, as expected, the rate of convergence of the GASS algorithms is faster since there is a larger step size update. In this case we can see that the Benveniste GASS LSM is as fast as the fast LSM. In general, Benveniste was always faster than Ang & Farhang, which in turn was faster than Matthews & Xie. We can also see the effect of ρ in the plots in the right side of the figures which show the progression of the step size for the three GASS algorithms. We see that in the case of a larger ρ the step sizes reach a higher value and are also more unstable. Both of these observations are expected since they stem from Equation 18.

Regarding steady state error variance, for $\rho = 0.001$, the standard LMS with lower step size has the lowest steady state variance. The Ang & Farhang and Matthews & Xie GASS LMS schemes are comparable and have slightly higher variance, and finally Benveniste GASS LMS and the standard LMS with $\mu = 0.1$ have the highest steady state error variance. These results are further enforced by examining the step sizes for the GASS LMS schemes in the right plot, where we again see that there is a positive correlation between step size and steady state error variance. In the case of $\rho = 0.005$, all GASS LMS schemes have increased steady state error variance, comparable to that of the standard LSM with $\mu = 0.1$.

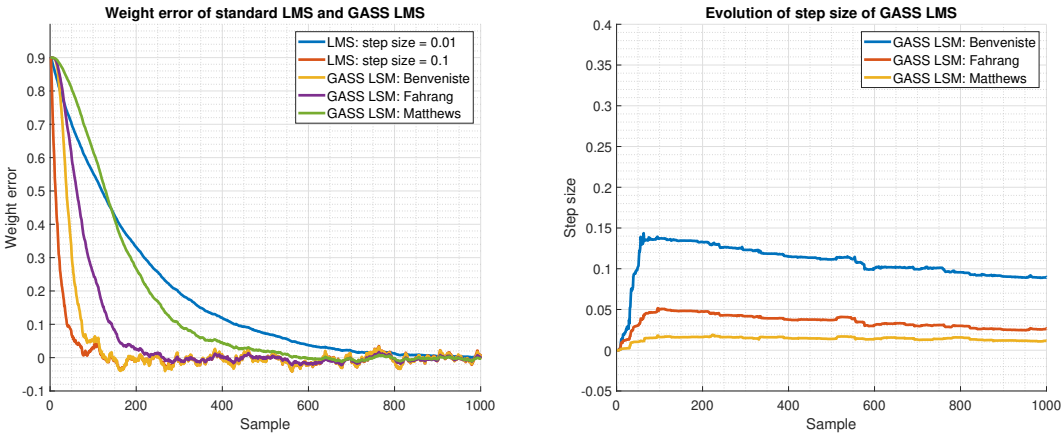


Figure 18: Evolution of weight error of LMS and different GASS LMS schemes, $\rho = 0.001$

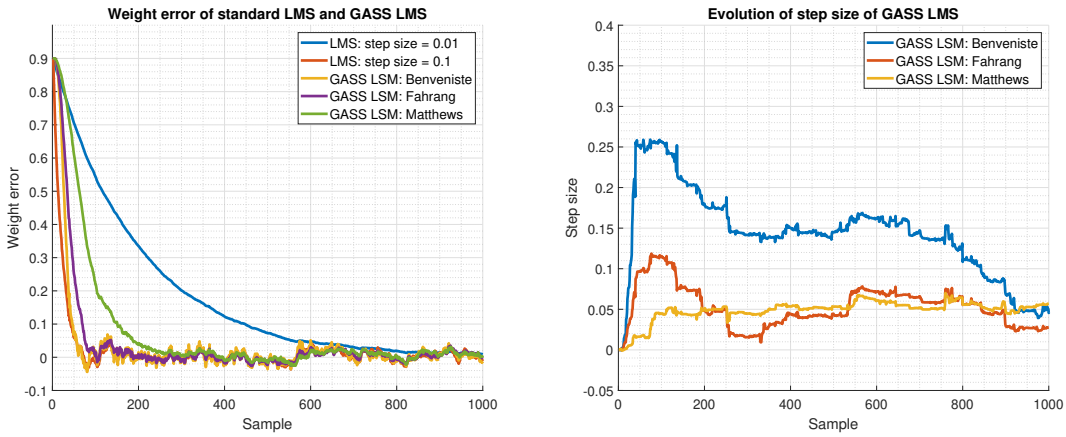


Figure 19: Evolution of weight error of LMS and different GASS LMS schemes, $\rho = 0.005$

As the weight update in the LMS is proportional to the input vector $\mathbf{x}(n)$, it is logical to want to normalise the step size. This is achieved through the Normalized LMS (NLMS) algorithm. We start from the update equation based on the a posteriori error $e_p(n)$.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \tag{2.24}$$

$$\begin{aligned} e_p(n) &= d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1) && (2.25)\\ &= d(n) - \mathbf{x}^T(n)\left(\mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n)\right) && (2.26)\\ &= d(n) - \mathbf{x}^T(n)\mathbf{w}(n) - \mu e_p(n)\|\mathbf{x}(n)\|^2 && (2.27)\\ &= e(n) - \mu e_p(n)\|\mathbf{x}(n)\|^2 && (2.28)\\ &= \frac{e(n)}{1+\mu\|\mathbf{x}(n)\|^2} && (2.29) \end{aligned}$$

Substituting within the weight equation we obtain the weight update as follows.

$$\begin{aligned} \Delta\mathbf{w}(n) &= \mathbf{w}(n+1) - \mathbf{w}(n) && (2.30)\\ &= \mu e_p(n)\mathbf{x}(n) && (2.31)\\ &= \mu\frac{e(n)}{1+\mu\|\mathbf{x}(n)\|^2}\mathbf{x}(n) && (2.32)\\ &= \frac{e(n)}{\frac{1}{\mu}+\|\mathbf{x}(n)\|^2}\mathbf{x}(n) && (2.33) \end{aligned}$$

By comparing with the normalized LMS (NLMS) update as shown in the equation below, we conclude that $\beta = 1$, and $\epsilon = \frac{1}{\mu}$.

$$\Delta\mathbf{w}_{NLMS}(n) = \frac{\beta e(n)}{\varepsilon + \|\mathbf{x}(n)\|^2}\mathbf{x}(n) \tag{2.34}$$

By making the regularisation factor ϵ time varying and adaptive through a gradient method we obtain the Generalized Normalized Gradient Descent (GNGD) algorithm. In Figure 20 below we compare the performance of GNGD to the Benveniste GASS LMS with regards to weight error for the MA(1) process mentioned above. Note that both algorithms had an initial step size of 0.05. Clearly, the GNGD algorithm converges much faster, reaching zero weight error at 20 samples, whereas the Benveniste GASS LSM algorithm achieves this in more

than 120 samples. That said, the steady state error variance is much better in the case of the GASS algorithm, which achieves an EMSE of 0.008, whereas the GNGD algorithms has an EMSE of 0.126.

Finally, The GNGD algorithm has much lower complexity than the Benveniste GASS algorithm. GNGD is of $O(N)$ complexity, since there are only inner products and additions for the updated of each time instant n, while Benveniste GASS is of complexity $O(N^2)$, caused by the outer product involved in the step size update. Therefore, GNGD can offer better convergence at a much lower computational complexity, although this comes at the price of steady state error variance.

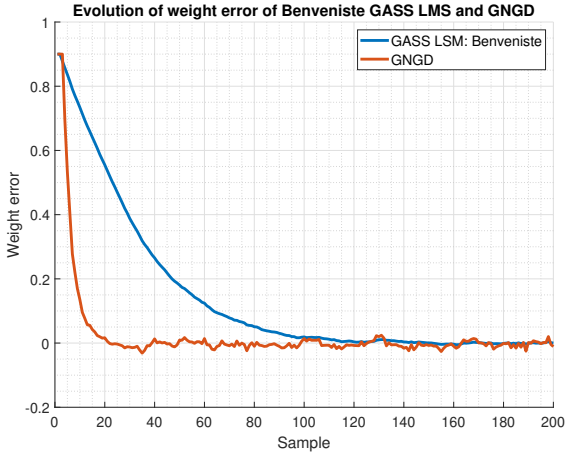


Figure 20: Evolution of weight error of Benveniste GASS LMS and GNGD schemes

2.3 Adaptive Noise Cancellations

We start from the expression for the MSE of the Adaptive Line Enhancer (ALE). Note that $\eta(n)$ in this case is coloured noise generated through an MA process according to the equation below, where $v(n)$ is white noise with unit variance.

$$\eta(n) = v(n) + 0.5v(n-2) \tag{2.35}$$

$$\begin{aligned}
\mathbb{E}\left\{(s(n)-\hat{x})^2\right\} &= \mathbb{E}\left\{(x(n)+\eta(n)-\hat{x}(n))^2\right\} && (2.36)\\
&= \mathbb{E}\left\{(x(n)-\hat{x}(n))^2\right\} + \mathbb{E}\left\{\eta^2(n)\right\} + 2\mathbb{E}\{(x(n)-\hat{x}(n))\eta(n)\} && (2.37)\\
&= \mathbb{E}\left\{(x(n)-\hat{x}(n))^2\right\} + \mathbb{E}\left\{\eta^2(n)\right\} + 2\mathbb{E}\{x(n)\eta(n)\} - 2\mathbb{E}\{\hat{x}(n)\eta(n)\} && (2.38)
\end{aligned}$$

The first term of the sum is not a function of noise. The second term of the sum is the noise power, which is constant. The third term is 0 since the signal $x(n)$ and the noise are uncorrelated. Therefore, the only term of interest is the fourth and final term. We expand this further below.

$$\mathbb{E}\left\{\hat{x}(n)\eta(n)\right\} = \mathbb{E}\left\{\mathbf{w}^T(n)\mathbf{u}(n)\eta(n)\right\} \tag{2.39}$$

$$= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k s(n-\Delta-k)\eta(n)\right\} \tag{2.40}$$

$$= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k (x(n-\Delta-k)+\eta(n-\Delta-k))\eta(n)\right\} \tag{2.41}$$

$$= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k x(n-\Delta-k)\eta(n)\right\} + \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k \eta(n-\Delta-k)\eta(n)\right\} \tag{2.42}$$

$$= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k \eta(n-\Delta-k)\eta(n)\right\} \tag{2.43}$$

$$= \mathbb{E}\left\{\sum_{k=0}^{M-1} w_k (v(n-\Delta-k)+0.5v(n-\Delta-k-2))(v(n)+0.5v(n-2))\right\} \tag{2.44}$$

Since, as mentioned, $v(n)$ is white i.i.d. noise, $\mathbb{E}\{v(n-k)v(n-l)\} = 0$ for k. Therefore, to minimise the MSE, we have to ensure no overlap, which occurs for $\Delta > 2$. So, $\Delta = 3$.
In Figure 21 we explore the effect of different delays on the performance of the LMS ALE algorithm in denoising a noisy sine. In the left side of the plot we see that as the delay rises from 1 through 4 the performance appears to be getting better. Our results are confirmed with the plot in the right side of the figure, where we explore the relationship between delay and mean square prediction error (MSPE) explicitly. We observe that for $\Delta < 2$ the MSPE is large. This is expected, since the algorithm is unable to capture the variations of the coloured noise from Equation 2.3. Our theoretical result is validated, as we see that the error is minimised for delay $\Delta = 3$. After this point, the error starts to increase. Again, this makes sense as the ALE does not take into account that the noise is correlated with the delayed versions of itself.

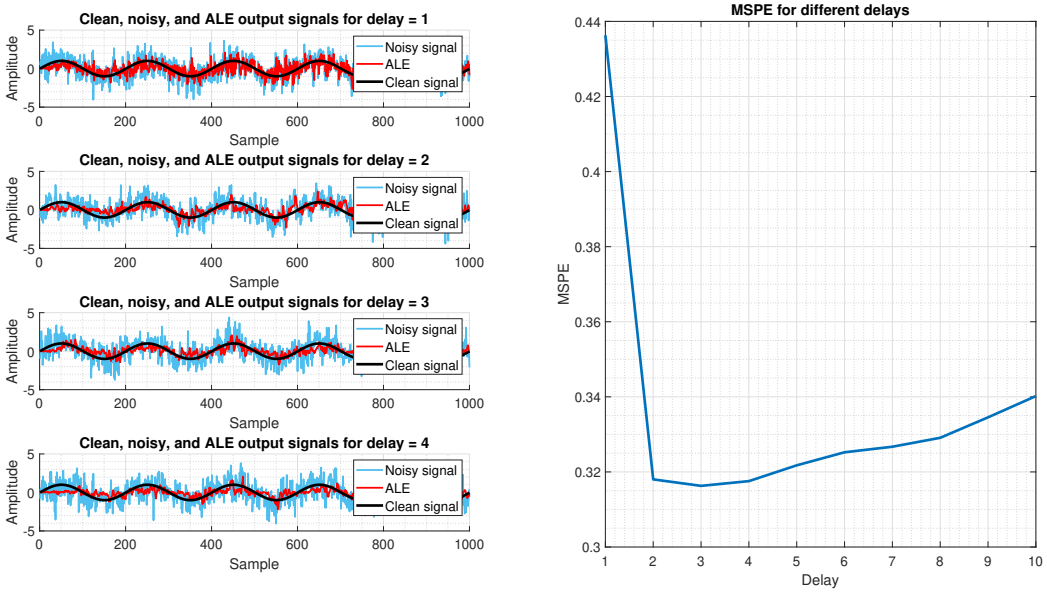


Figure 21: Output (left) and error (right) of ALE with LMS for different values of delay

Figure 22 again investigates the effect of the delay, as well as of the model order, on the mean square prediction error (MSPE). The same explanation for the delay effect holds as in the previous figure. Regarding model order, we surprisingly observe the it is also positively correlated with increased MSPE. We can explain this through

the fact that the larger model orders allow the ALE to overfit, meaning that it considers the noise as part of the signal and fits it. As the computational complexity also increases with larger model orders, we would suggest a choice of $order = 5, delay = 3$.

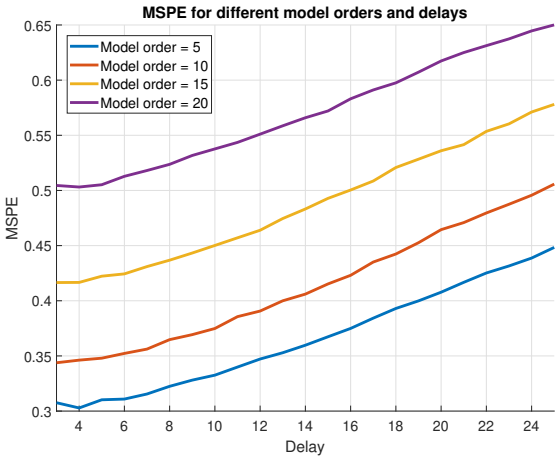


Figure 22: MSPE of ALE with LMS for different model orders and delays

In Figure 23 we have implemented a variation of the ALE, named Adaptive Noise Cancellation (ANC), which accepts a secondary noise input correlated in some unknown way with the primary noise. Here we compare the performance of the two algorithms in denoising the noisy sine. Visually, the ANC scheme has much better performance than the ALE algorithm. This is validated over our experiment, where we ran 1000 independent trials using the optimal configuration from above (i.e. $order = 5, delay = 3$) for both algorithms. The MSPE of ALE was measured to be $MPSE_{ALE} = 0.1158$, whereas the corresponding error of ANC was $MPSE_{ANC} = 0.0042$, a result over 27 times better!

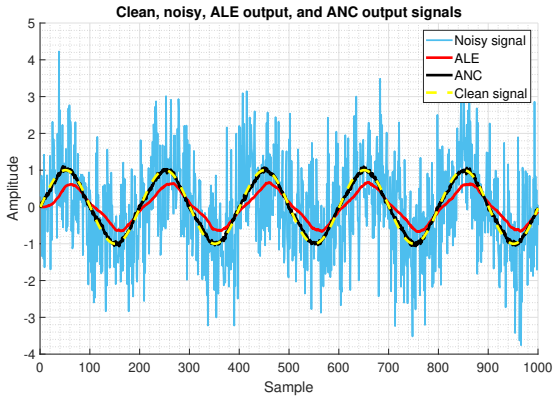


Figure 23: Output estimates of ALE and ANC with LMS

Finally, we test the performance of ANC in removing the mains interference from a EEG POz signal. In Figure 24 we plot the spectrograms for different choices of the model order M and step size μ. In this case performance is judged by the ability of the configuration to suppress the interference without altering the signal component itself. We can observe that for a particular value of step size, increasing the model order removes the noise more and more heavily. This is not always desirable, however, since in some cases, such as $M = 20, \mu = 0.01$ for example, ANC removes parts of the white noise and of the signal component itself as well. In other cases where the order and step size are both small, such as $M = 5, \mu = 0.0001$ for example, we can see that the interference is only partially removed. Optimal denoising appears to happen for $M = 10, \mu = 0.001$, where the interference is completely removed without altering the remaining spectrum.

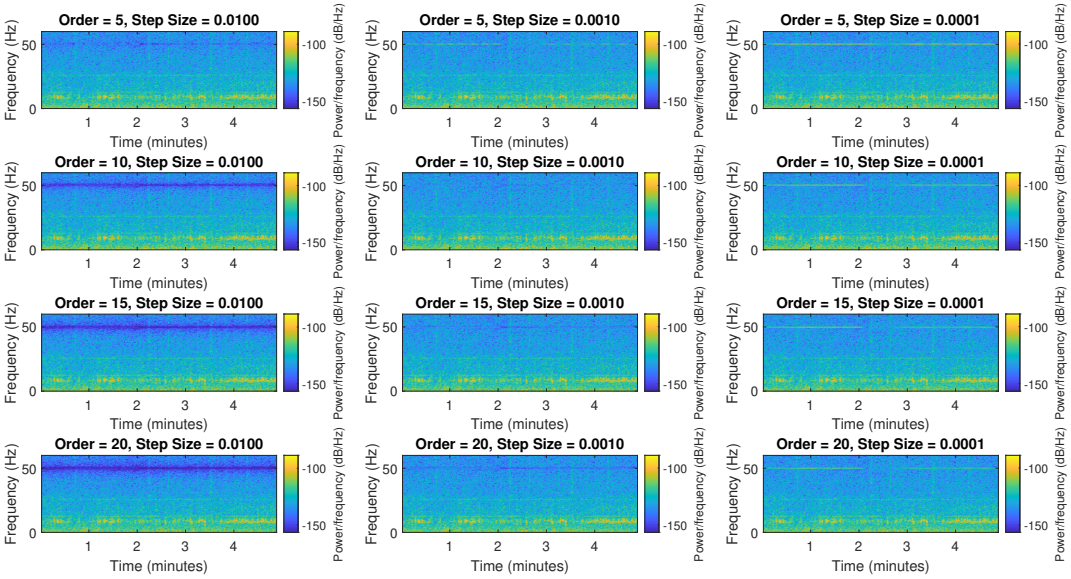


Figure 24: Spectrograms of denoised ANC output for different model orders and step sizes

In Figure 25 we plot the original spectrogram of the POz signal together with the optimally denoised spectrogram. As mentioned, optimal denoising was chosen to be at $M = 10, \mu = 0.001$. Clearly, the 50Hz intereference is removed without removing the desirable parts of the signal. For example, the tiredness-induced alpha-rhythm (8-10Hz), SSVEP (13Hz), and harmonics of the SSVEP (26, 39Hz) are still clearly visible. These results are validated in the plot under the two spectrograms, where we depict a spectrum including the periodograms of both the original and the denoised version. We observe that ANC in the current configurations has managed to retain the overall spectrum while suppressing only the 50Hz interference.

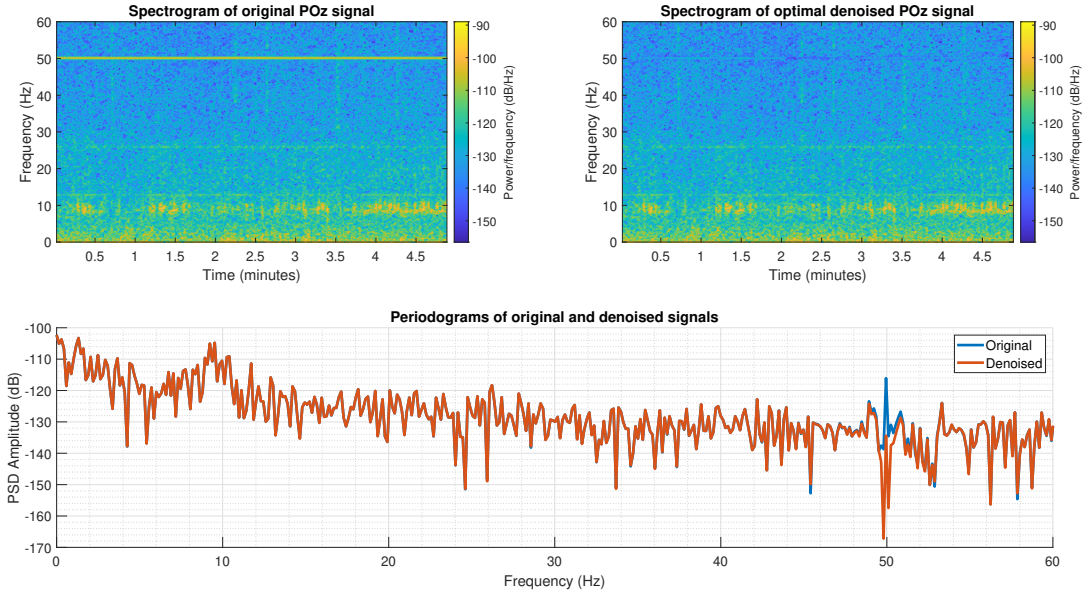


Figure 25: Spectrograms of original POz signal (left) and denoised ANC output (right) and corresponding periodograms (bottom)

3 Widely Linear Filtering and Adaptive Spectrum Estimation

3.1 Complex LMS and Widely Linear Modelling

In this section we extend the work of the previous section by focusing on complex signals. We start the analysis of this section by defining the widely-linear-moving-average process, WLMA(1) as seen in the equation below. Note that $x(n)$ in this case refers to circular white Gaussian noise. Also note that $b_1 = 1.5 + 1j$ and $b_2 = 2.5 - 0.5j$.

$$y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-1) \tag{3.1}$$

To deal with such complex signals we introduce two new algorithms which are extensions of the standard LMS scheme: Complex LMS (CLMS) and Augmented Complex LMS (ACLMS), as shown below. As can be seen ACLMS operates using an augmented input $\mathbf{x}_a = \begin{bmatrix} \mathbf{x}^T & \mathbf{x}^H \end{bmatrix}^T$ and two sets of weights, $\mathbf{h}(n)$ and $\mathbf{g}(n)$. These changes are employed to the widely linear model in order to equip it with sufficient degrees of freedom to fully exploit the second order statistics in the data.

CLMS

$$\begin{aligned} y(n) &= \mathbf{h}^H(n)\mathbf{x}(n) \\ e(n) &= y(n) - \hat{y}(n) \\ \mathbf{h}(n+1) &= \mathbf{h}(n) + \mu e^*(n)\mathbf{x}(n) \end{aligned}$$

ACLMS

$$\begin{aligned} \hat{y}(n) &= \mathbf{h}^H(n)\mathbf{x}(n) + \mathbf{g}^H(n)\mathbf{x}^*(n) \\ e(n) &= y(n) - \hat{y}(n) \\ \mathbf{h}(n+1) &= \mathbf{h}(n) + \mu e^*(n)\mathbf{x}(n) \\ \mathbf{g}(n+1) &= \mathbf{g}(n) + \mu e^*(n)\mathbf{x}^*(n) \end{aligned}$$

The upper side of figure 26 depicts the circularity plots for the $x(n)$ (WGN) and $y(n)$ (WLMA) signals. Clearly, the WLMA signal is non-circular, as opposed to the WGN signal. It's important to note that the expectation of the circularity coefficient for the WGN is 0, and that the offset in this case is most likely caused by a small sample size. In the bottom plot we depict the learning curves of the CLMS and ACLMS algorithms in adaptively estimating the WLMA signal. Evidently, that CLMS algorithm is unable to capture the second order statistics of the WLMA signal, exhibiting large error throughout the range of the signal. ACLMS on the contrary manages to model the signal accurately by using the extra degree of freedom.

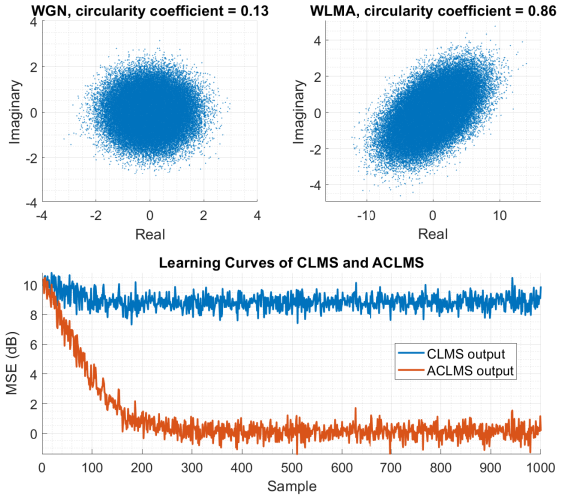


Figure 26: Circularity plots for WGN and WLMA signals (top) and learning curves of CLMS and ACLMS predictors for WLMA signal

Next, we apply the CLMS and ACLMS algorithms to real-world wind data. In the left side of Figure 27 we show the circularity plots for the three different wind regimes: low wind, medium wind, and high wind. As can be seen from the plots, the low wind is the most circular while high wind is the least circular. In the right side of the figure, for each of the wind regimes, we have plotted the performance of the CLMS and ACLMS algorithms

in terms of MSE as a function of model order. The step sizes chosen for low, medium, and high wind are 0.2, 0.02, and 0.002 respectively. In general, all algorithms exhibit convex error functions and clear minima with respect to model order. Additionally, ACLMS appears to produce the lowest MSE values overall. The medium wind case is an exception, since CLMS and ACLMS appear to perform equally well for some model orders.

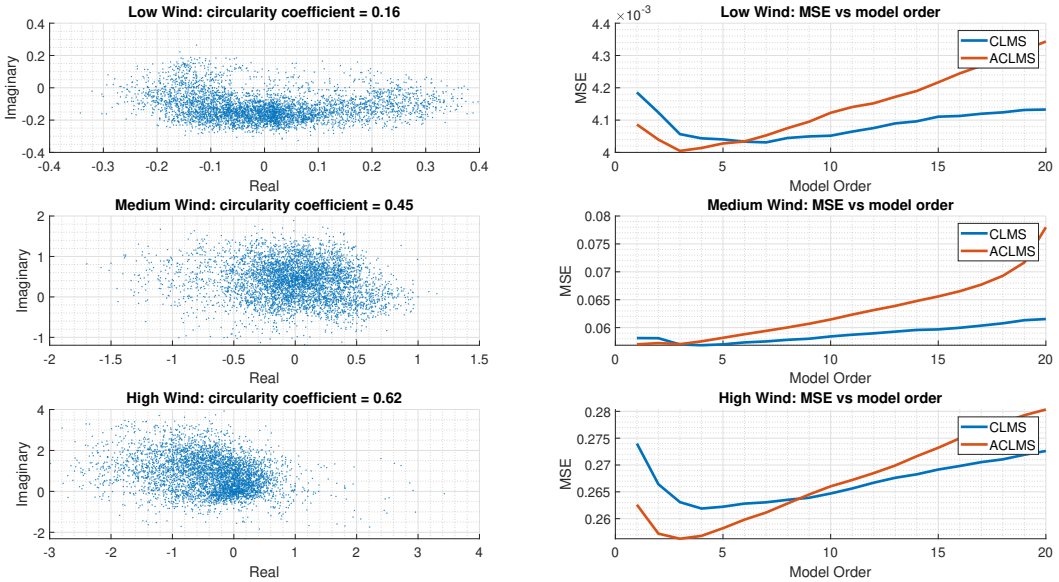


Figure 27: Circularity plot (left) and one-step ahead prediction using CLMS and ACLMS (right) for different complex wind profiles

Next, we apply the same concepts in the context of three-phase power systems. In general, three-phase systems are characterised by three sinusoidal voltage signals of different amplitudes and phases, $v_a, v_b,$ and v_c. In a balanced operating condition all three signals have the same peak voltage and are spaced equally apart in phase, i.e. the three voltages have a phase shift of exactly 120 degrees. Unbalanced refers to any operating setting where these conditions are not exactly met. By applying the Clarke $(\alpha - \beta)$ transform, the three-phase system can be projected onto two orthogonal axes v_α and v_β. Subsequently, the complex clarke $(\alpha - \beta)$ voltage $v(n) = v_\alpha(n) + jv_\beta(n)$ can be obtained. Figure 28 depicts the circularity plots for the clarke $(\alpha - \beta)$ voltages balanced and unbalanced three-phase systems. We observe that the balanced case is perfectly circular whereas the unbalanced case is non-circular.

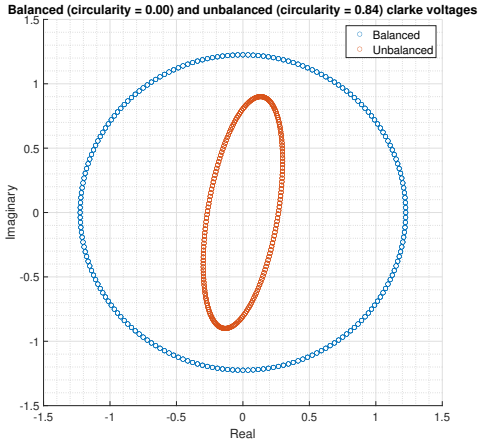


Figure 28: Circularity plot for balanced and unbalanced clarke voltages

We aim to apply CLMS and ACLMS for estimating the system frequency of the system. The sequence of equations below aims to find an expression for the frequency in terms of the weight vectors of the CLMS and ACLMS schemes. We start with the clarke $(\alpha - \beta)$ voltage in the balanced case for the strictly-linear model:

$$v(n) = \sqrt{\frac{3}{2}} V e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} \tag{3.2}$$

$$v(n+1) = h^*(n) v(n) \tag{3.3}$$

Therefore the conjugate of the strictly-linear weight vector is given by the following:

$$h^*(n) = \frac{v(n+1)}{v(n)} \tag{3.4}$$

$$= \frac{\sqrt{\frac{3}{2}} V e^{j\left(2\pi \frac{f_0}{f_s}(n+1) + \phi\right)}}{\sqrt{\frac{3}{2}} V e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)}} \tag{3.5}$$

$$= \frac{\sqrt{\frac{3}{2}} V e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} e^{j 2\pi \frac{f_0}{f_s}}}{\sqrt{\frac{3}{2}} V e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)}} \tag{3.6}$$

$$= e^{j 2\pi \frac{f_0}{f_s}} \tag{3.7}$$

We obtain the weight vector by taking the conjugate and equating the magnitude and phase components to obtain the expression for the system frequency:

$$h(n) = |h(n)| e^{j\left(\arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right)\right)} = e^{-j 2\pi \frac{f_0}{f_s}} \tag{3.8}$$

$$-2\pi \frac{f_o}{f_s} = \arctan\left(\frac{\Im(h(n))}{\Re(h(n))}\right) \tag{3.9}$$

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

Next we examine the unbalanced case for the widely-linear model, and use the assumption that amplitude changes from adjacent samples are negligible.

$$v(n) = A(n) e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} + B(n) e^{-j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} \tag{3.11}$$

$$v(n+1) = A(n+1) e^{j\left(2\pi \frac{f_0}{f_s}(n+1) + \phi\right)} + B(n+1) e^{-j\left(2\pi \frac{f_0}{f_s}(n+1) + \phi\right)} \tag{3.12}$$

$$\approx A(n) e^{j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} e^{j 2\pi \frac{f_0}{f_s}} + B(n) e^{-j\left(2\pi \frac{f_0}{f_s} n + \phi\right)} e^{-j 2\pi \frac{f_0}{f_s}} \tag{3.13}$$

From the widely linear model we get the update as follows:

$$v(n+1) = h^*(n) v(n) + g^*(n) v^*(n) \tag{3.14}$$

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

By equating Equations 3.13 and 3.15 we obtain the following:

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

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

$$\Rightarrow \quad e^{j\left(2\pi\frac{f_0}{f_s}\right)} = h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} \tag{3.18}$$

$$\Rightarrow \quad e^{-j\left(2\pi\frac{f_0}{f_s}\right)} = h^*(n) + g^*(n)\frac{A^*(n)}{B(n)} \tag{3.19}$$

Therefore, since the two equations are complex conjugates:

$$h^*(n) + g^*(n)\frac{B^*(n)}{A(n)} = h(n) + g(n)\frac{A(n)}{B^*(n)} \tag{3.20}$$

$$g^*(n)\left(\frac{B^*(n)}{A(n)}\right)^2 + (h^*(n) - h(n))\left(\frac{B^*(n)}{A(n)}\right) - g(n) = 0 \tag{3.21}$$

We solve the quadratic w.r.t $\frac{B^*(n)}{A(n)}$ to obtain the solutions:

$$\frac{B^*(n)}{A(n)} = \frac{-\left(h^*(n) - h(n)\right) \pm \sqrt{\left(h^*(n) - h(n)\right)^2 + 4g^*(n)g(n)}}{2g^*(n)} \tag{3.22}$$

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

Substituting the $\frac{B^*(n)}{A(n)}$ term from Equation 3.18:

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

$$= \Re(h(n)) \pm j\sqrt{\Im(h(n))^2 - |g(n)|^2} \tag{3.25}$$

Finally, by equating the phase and keeping the positive (+) version since $fs > fo > 0$, we obtain the expression for the system frequency:

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

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

In Figure 29 we depict the performance of the CLMS and ACLMS algorithms in estimating the frequency in balanced and unbalanced systems. In the balanced case where the clarke $(\alpha-\beta)$ voltage is circular both CLMS and ACLMS converge to the correct frequency. That said, ACLMS has a much lower convergence speed than CLMS.

In the unbalanced case, where the clarke $(\alpha - \beta)$ voltage is non-circular, while the ACLMS converges to the correct system frequency, CLMS does not, instead giving an estimate of 21Hz. Just as in the WLMA case explored above, this can be explained through the fact that CLMS only has one degree of freedom and is therefore unable to model the non-circular data.

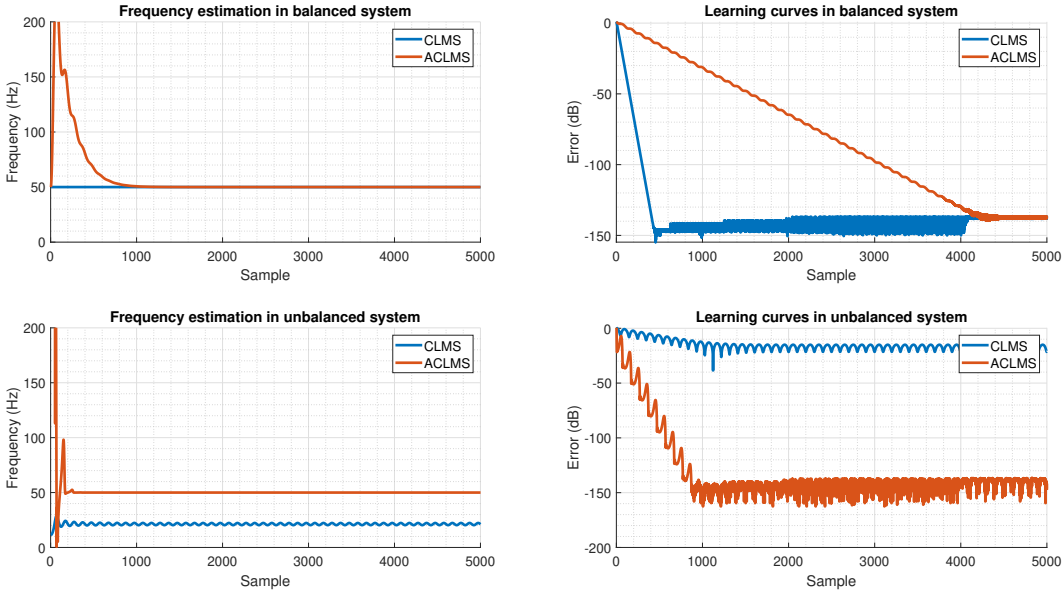


Figure 29: Frequency estimation using CLMS and ACLMS schemes (left) and corresponding learning curve (right) for balanced (top) and unbalanced (bottom) systems

3.2 Adaptive AR Model Based Time-Frequency Estimation

In this section we explore the concepts of non-stationary (online) spectrum estimation via the complex adaptive schemes mentioned previously. We begin by defining the FM signal $y(n)$ which we will use as the input in the following experiments.

$$y(n) = e^{j\left(\frac{2\pi}{f_s}\phi(n)\right)} + \eta(n) \tag{3.28}$$

$$\phi(n) = \int f(n)dn \tag{3.29}$$

$$f(n) = \begin{cases} 100, & 1 \leq n \leq 500 \\ 100 + \frac{n-500}{2}, & 501 \leq n \leq 1000 \\ 100 + \left(\frac{n-1000}{25}\right)^2, & 1001 \leq n \leq 1500 \end{cases} \tag{3.30}$$

In the left side of figure 30 below we have plotted the frequency variation and the corresponding phase of the FM signal. In the right side of the figure we examine the performance of using the traditional AR-based approaches as presented in Section 1.4. We observe that the resulting PSD does not capture the frequency variation of Equation 3.30, instead showing a single peak around 180Hz. This can be explained through the fact that the traditional methods do not account for the non-stationarity of the FM signal.

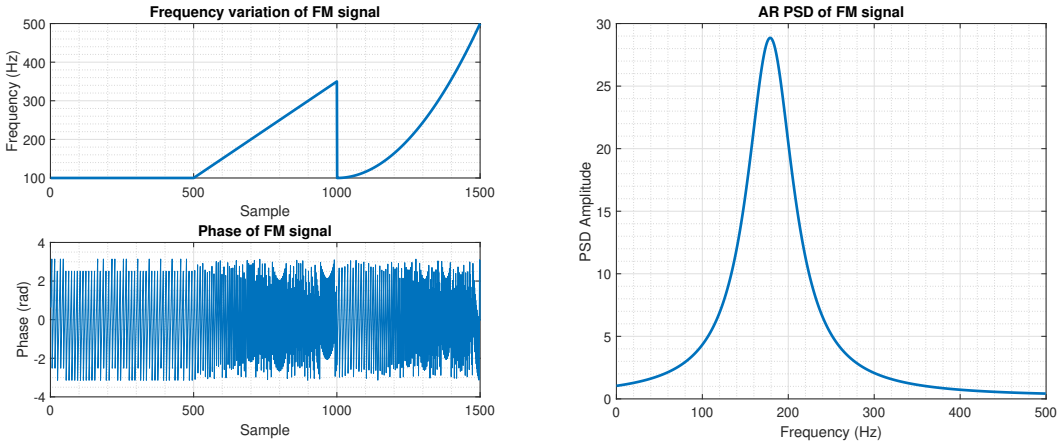


Figure 30: Frequency variation and corresponding phase of FM signal (left) and PSD estimation from AR(1) process (right)

We now attempt to estimate the AR coefficient of $y(n)$ adaptively through the use of the CLMS algorithm. Figure 31 depicts the performance of CLMS in producing the time-frequency spectrum of the FM signal. We can observe that for a correct value of the step size (in this case $\mu = 0.1$), the adaptive approach performs better than the block-based approach since it is able to capture the changes in frequency. For small step sizes, however, the algorithm does not converge quickly enough to produce the correct variations in frequency. On the other hand, large step sizes introduce larger error variance. Again we observe the tradeoff between these two quantities.

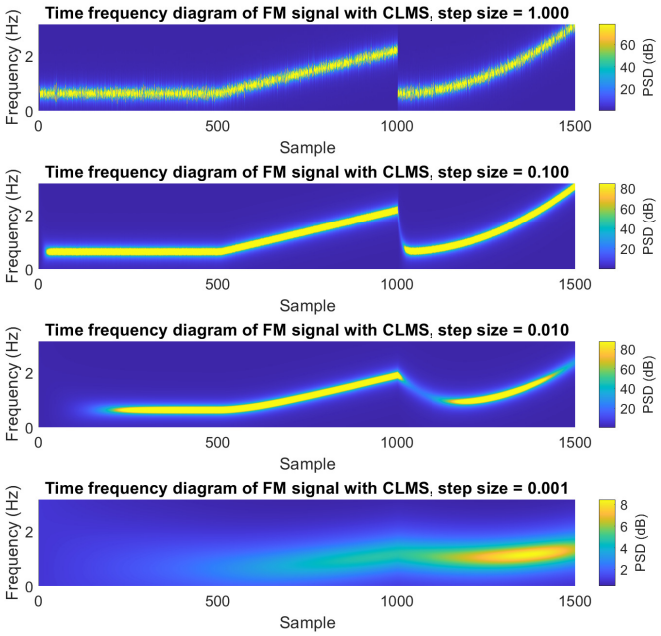


Figure 31: Time frequency diagram of FM signal using CLMS(1) and different step sizes

3.3 A Real Time Spectrum Analyser Using Least Mean Square

This section begins with mathematical exploration of the estimation of signal $y(n)$ as a linear combination of N harmonically related sinusoids as shown below. The relationship is also given in matrix-vector form in Equation 3.32.

$$\hat{y}(n) = \sum_{k=0}^{N-1} w(k) e^{j2\pi kn/N} \tag{3.31}$$

$$\hat{\mathbf{y}} = \mathbf{F}\mathbf{w} \tag{3.32}$$

In general, the least squares scheme aims to find $\mathbf{w}$ by minimising the sum of squared errors between the estimated signal $\hat{\mathbf{y}}$ and true signal $\mathbf{y}$ as shown below:

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

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

$$= \min_{\mathbf{w}} \left\{ \mathbf{y}^H\mathbf{y} - \mathbf{w}^H\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}\mathbf{w} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w} \right\} \tag{3.35}$$

To minimise the cost function, we take the derivative with respect to the weights vector $\mathbf{w}$.

$$\frac{dC(\mathbf{w})}{d\mathbf{w}} = \frac{d}{d\mathbf{w}} \left(\mathbf{y}^H\mathbf{y} - \mathbf{w}^H\mathbf{F}^H\mathbf{y} - \mathbf{y}^H\mathbf{F}\mathbf{w} + \mathbf{w}^H\mathbf{F}^H\mathbf{F}\mathbf{w} \right) \tag{3.36}$$

$$= -2\mathbf{F}^H\mathbf{y} + 2\mathbf{F}^H\mathbf{F}\mathbf{w} \tag{3.37}$$

Equating the derivative to 0 to find the optimum we obtain the expression for the least square solution for the weights vector $\hat{\mathbf{w}}$. Note that this assumes that the matrix $\mathbf{F}^H\mathbf{F}$ is invertible.

$$\hat{\mathbf{w}} = \left(\mathbf{F}^H\mathbf{F}\right)^{-1} \mathbf{F}^H\mathbf{y} \tag{3.38}$$

In our case the matrix $\mathbf{F}$ from equation 3.32 is the Inverse Discrete Fourier Transform (IDFT) matrix, which is unitary. Thus, $\left(\mathbf{F}^H\mathbf{F}\right)^{-1} = \mathbf{I}$ and equation 3.38 becomes:

$$\mathbf{y} = \left(\mathbf{F}^H\right)^{-1} \hat{\mathbf{w}} = \mathbf{F}\hat{\mathbf{w}} \tag{3.39}$$

This is the DFT formula. Therefore, the DFT can be seen as a projection of a vector (signal) from the time domain to the frequency domain, the basis of which is defined by the column vectors of F, which are harmonically related sinusoids.
By treating the DFT as a least squares problem we now aim to implement an adaptive version by using the CLMS. In this case, the desired signal of the CLMS will be the time domain signal and the input signal at time instant n will be the complex phasor $\mathbf{x}(n)$ as given below:

$$\mathbf{x}(n) = \frac{1}{N} \left[1, \quad e^{j\frac{2n\pi}{N}}, \quad e^{j\frac{4n\pi}{N}}, \quad \cdots \quad , \quad e^{j\frac{2n(N-1)\pi}{N}} \right]^T \tag{3.40}$$

For this variation of the CLMS the estimated output $\hat{y(n)}$ is the inner product of the weight vector with the entire input $\mathbf{x}(n)$, rather than a slice of it, as in previous versions. By making this change, the CLMS weights will attempt to adapt and approximate the DFT coefficients. Note that this occurs for $\mu = 1$ and $\mathbf{w}(0) = \mathbf{0}$. In Figure 32 we apply this algorithm to the FM signal mentioned above. By plotting the weight vector at each time step, we obtain a time-frequency diagram. However, we see that the CLMS-DFT algorithm does not produce the expected results. Instead, we see that the weights seem to increase cumulatively, retaining all past values. Since the expected result would require "forgetting" the past values of the weights, then this appears as an optimal setting to add leakage to the algorithm. Indeed, by adding a leakage coefficient and finetuning to "forget" the right amount we obtain the correct plot for a leakage coefficient $\gamma = 0.1$.

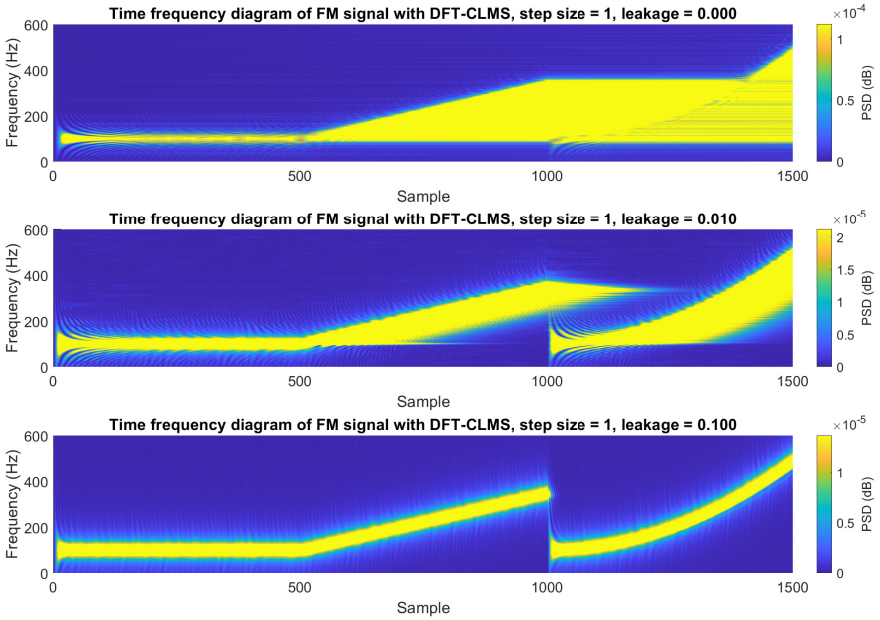


Figure 32: Time frequency diagram of FM signal using DFT-CLMS with $\mu = 1$ and different values of leakage

Finally, we apply the DFT-CLMS algorithm in order to produce a time-frequency diagram for an EEG POz signal. The results of this process are depicted in Figure 33. In general, DFT-CLMS was able to capture several of the frequency characteristics of the POz signal accurately. However, results appeared to be highly dependent on the specific signal slice under investigation. By signal slice we refer to the 1200-sample segment selected from the original 354000-sample POz signal. For example, the top plot in the figure below starts from sample 1500. In this plot, DFT-CLMS was able to adaptively detect the the tiredness-induced alpha-rhythm (8-10Hz), the harmonics of the SSVEP (26, 39Hz), as well as the mains interference (50Hz). However, it failed to capture the actual SSVEP itself (13Hz). On the contrary, the bottom plot of the figure, starting at sample 3000, accurately captures the SSVEP (13Hz), one SSVEP harmonic (26Hz), and a vague alpha-rhythm (8-10Hz), but was unable to capture the 50Hz interference. That said, it is still impressive that DFT-CLMS is able to capture all of this frequency information adaptively.

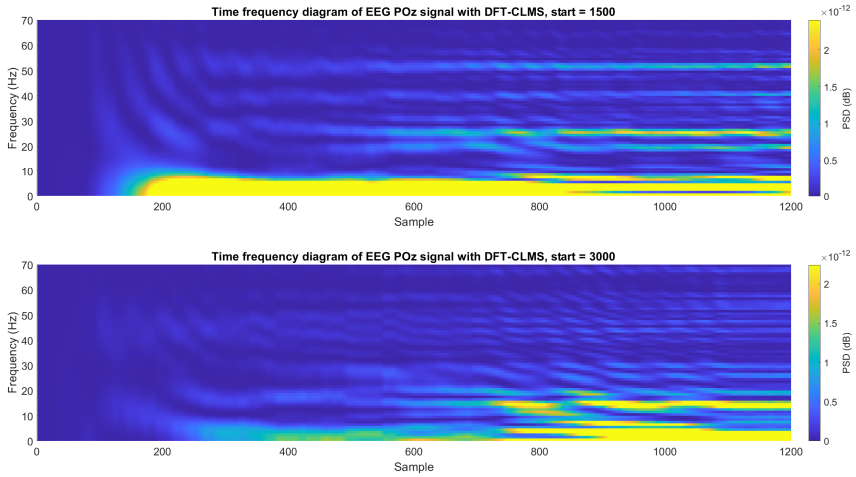


Figure 33: Time frequency diagram of EEG POz signal using DFT-CLMS with $step_size = 1$ and different values of leakage

4 From LMS to Deep Learning

In this section we further investigate the power of the LMS in modelling non-stationary data and iteratively add more complexity to the algorithm, with a route towards the popular deep learning perceptron model. We start by applying the standard LMS algorithm with $order = 4$, $\mu = 1 * 10^{-5}$ to the time series $y(n)$ after removing its mean. The output of the LMS along with the original time series are depicted in Figure 34. We observe that after around 200 samples the LMS is able to model $y(n)$ relatively well, meaning that it can capture the trend of the data despite having consistently smaller amplitude. In this case the MSE is equal to $MSE = 16.03dB$ and the prediction gain is $R_p = 5.197dB$.

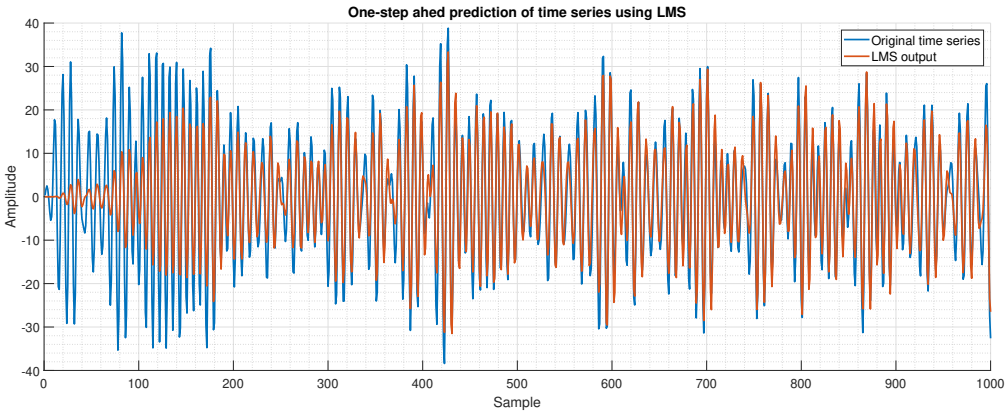


Figure 34: Zero mean time series and output of one-step ahead prediction using LMS

As a second step, we add a non-linear activation function (in this case a *tanh* function) to the output of the LMS. Again, we attempt to model the time series $y(n)$ with removed mean. From Figure 35 we can see that while the output appears to capture the frequency of the signal well, its amplitude is capped, leading to a very large MSE ($MSE = 22.94dB$) and a very low prediction gain ($R_p = -23.19dB$). Naturally, this makes sense, as the *tanh* function is bounded between $[-1, 1]$ for all values.

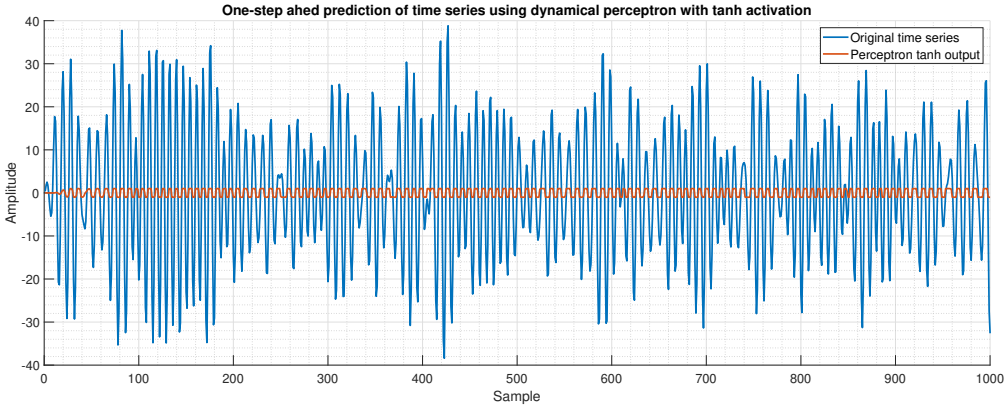


Figure 35: Zero mean time series and output of one-step ahead prediction using a dynamical perceptron with a *tanh* activation function

To battle the aforementioned issue we add a scaling factor a to the activation function ($a * tanh$). To reason about appropriate values for a we observe that the amplitude of the time series after having removed its mean goes up to almost 40. In this light, and since the *tanh* function is normally bounded between $[-1, 1]$, the scaling factor should be at least 40 in order to be able to capture the entire range of the signal. Figure 36 depicts the output of the dynamical perceptron for different values of the scaling factor. As expected, $a = 10$ is unable to capture the full dynamic range. Additionally, $a = 70$ appears to perform better than $a = 40$.

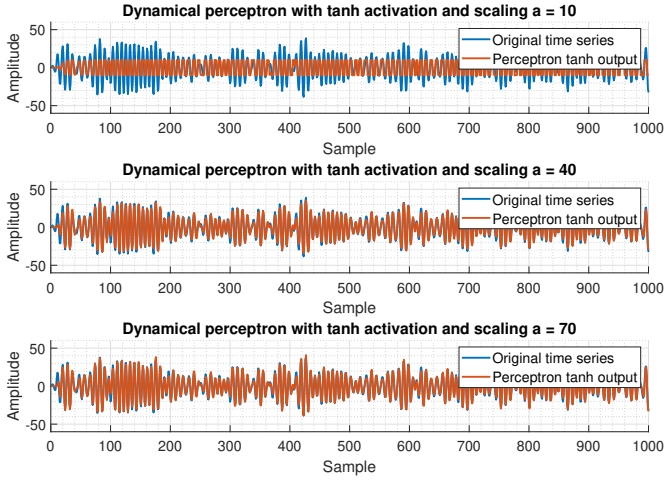


Figure 36: Zero mean time series and output of one-step ahead prediction using a dynamical perceptron with a *tanh* activation function for different values of scaling factor a

To investigate this further we plot the MSE and prediction error for $1 \leq a \leq 100$ in Figure 37. We observe that at $a = 81$ we obtain both the minimum MSE ($6.79dB$), as well as the maximum prediction gain ($16.5dB$). Therefore, choosing $a = 81$ we obtain the optimised scaling output in Figure 38, which achieves the MSE and prediction gain mentioned above. Therefore, this version of the algorithm performs better than the standard LMS scheme.

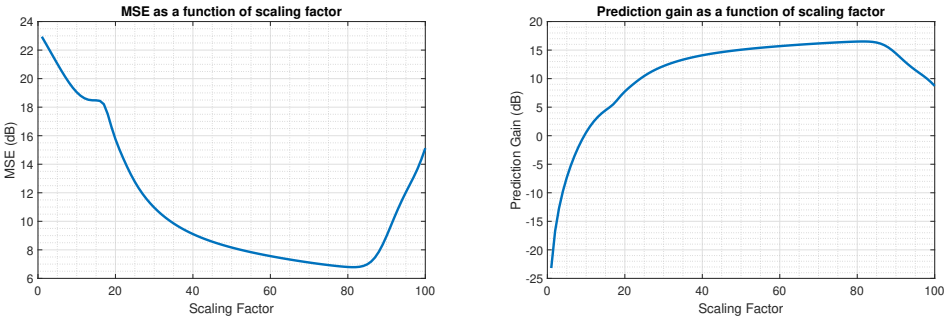


Figure 37: Prediction MSE (left) and prediction gain (right) as a function of the scaling factor a

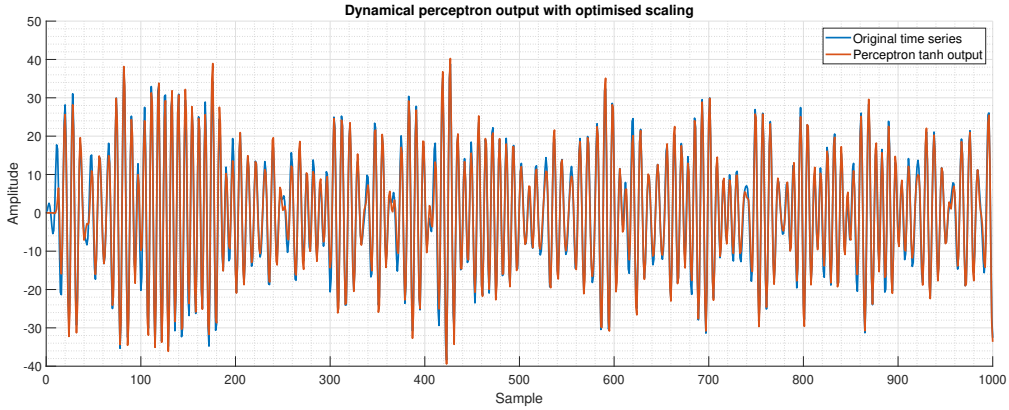


Figure 38: Zero mean time series and output of one-step ahead prediction using a dynamical perceptron with a *tanh* activation function and optimised scaling factor a

We now consider the time series without removing the mean. In order to account for a non-zero mean in our model we add a bias to the estimation prior to passing it through the activation function. Mathematically, given an input $\mathbf{x}$, a weight vector $\mathbf{w}$, and the activation function ϕ, we add the bias b as given by $\phi\left(\mathbf{w}^T\mathbf{x} + b\right)$. From the implementation perspective, by adding an additional weight to the weight vector and augmenting the input by padding with a single 1 at the beginning, as given by $\mathbf{x}_{augmented} = [1, \mathbf{x}]^T$, the model attempts to learn the bias as one of the weights.

Once again, we aim to find the optimal scaling factor for this setting by plotting the MSE and prediction error for $1 \leq a \leq 100$. This is depicted in Figure 39. Interestingly, in this case the scaling factor which attains both the minimum MSE ($11.28dB$), as well as the maximum prediction gain ($12.32dB$) is $a = 52$. Therefore, choosing $a = 52$ we obtain the optimised scaling output for the learnt bias model in Figure 40. This achieves the MSE and prediction gain mentioned above. We observe that approximately in the first 50 samples the perceptron output is nearly centered around 0 as the algorithm has still not learnt the bias in order to offset the mean. After around 150 samples the algorithm has learnt the bias relatively well and is able to model the data with reasonable accuracy. However, the model seems to approximate the data with consistently smaller amplitude than the original time series, leading to a persistent error on top.

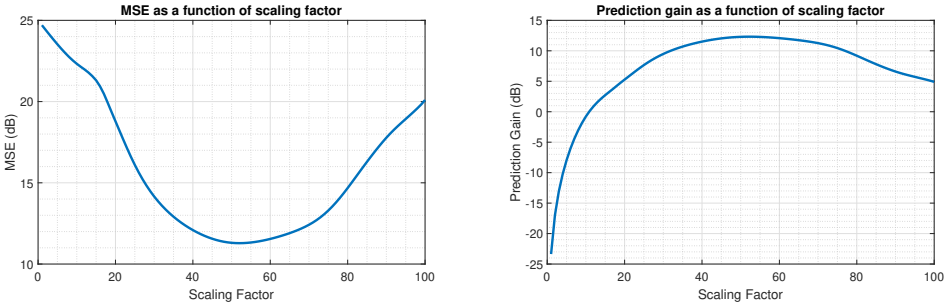


Figure 39: Prediction MSE (left) and prediction gain (right) as a function of the scaling factor a

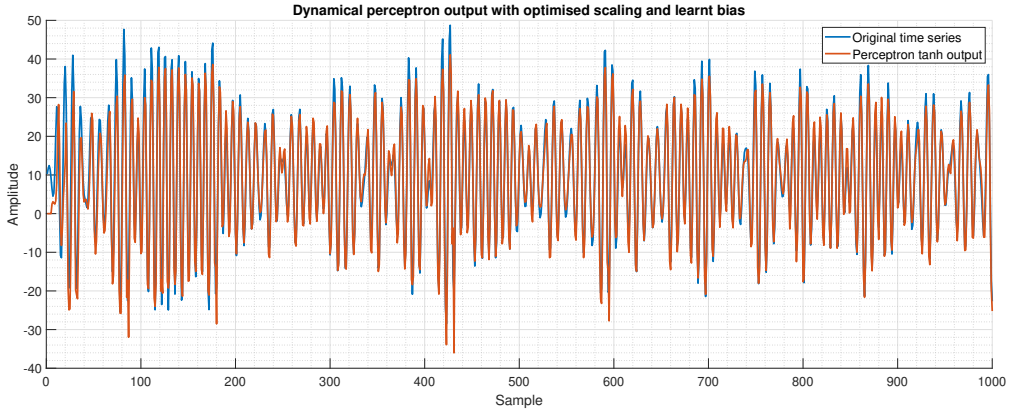


Figure 40: Original time series and output of one-step ahead prediction using a dynamical perceptron with a *tanh* activation function, optimised scaling factor a, and learnt bias

In order to speed up convergence, we pre-train the weights on the first 20 samples for 100 epochs. We subsequently use the resulting weights of this process as the initial weights of the perceptron, rather than the 0 vector we were previously using. The results are depicted in Figure 41. We observe that pretraining greatly improves both convergence speed and general performance. The algorithm now converges between 5-10 samples rather than 50+ samples without pretraining. Performance metrics are also improved, since the pretrained version achieves an MSE of $MSE = 6.60dB$ and a prediction gain of $R_p = 16.88dB$.

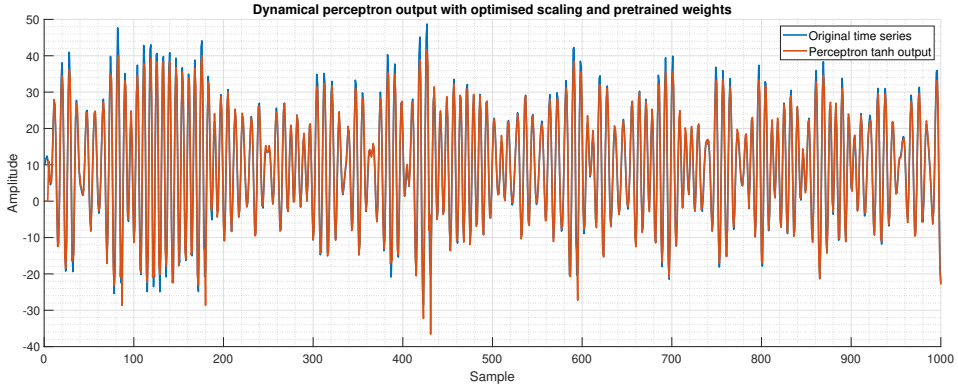


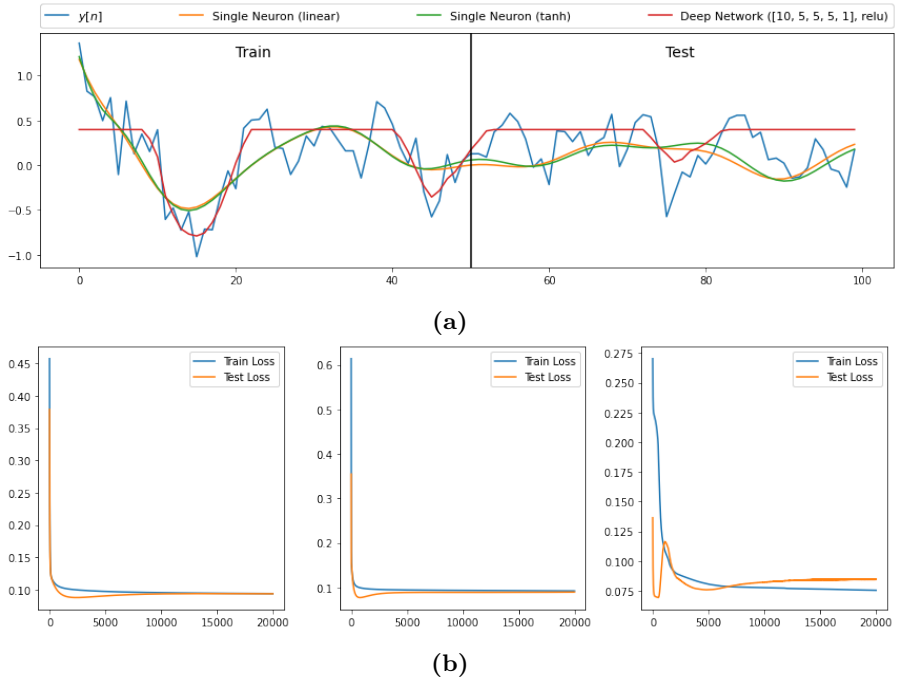
Figure 41: Original time series and output of one-step ahead prediction using a dynamical perceptron with a *tanh* activation function, optimised scaling factor a, and pretrained weights

By arranging several dynamical perceptrons together it is possible to create a deep network. The way in which weights are updated within such deep networks is through the error backpropagation algorithm. To understand why backpropagation exists and how it operates it is important to first provide a brief overview of the fundamental operating principles behind deep networks. As aforementioned, deep networks are constructed by placing several perceptrons in a particular configuration. The input is fed to the perceptrons of the first layer, the input layer. Then, after the perceptrons of this layer go through the operations mentioned above and described through the equation $\phi\left(\mathbf{w}^T\mathbf{x}+b\right)$, their output is passed as the input to the perceptrons of the next layer. The repetition of this process until the output is reached is named the "feedforward" phase. Through the network output, as well as the corresponding correct label (since backpropagation is applied in supervised settings), the error function produces a magnitude of the error for the prediction or estimation.

The role of backpropagation is to propagate the calculated error backwards throughout the network in order to adjust the network parameters (i.e. the weights and biases), with the aim of minimising the error at the output. The main concept behind this algorithm is that by using the chain rule the error can be iteratively traced back to each network parameter in order to compute its contribution to the error, and therefore, the amount it should change in order to minimise it. The chain rule is used since the contribution is quantified through the derivative of the error function with respect to a particular parameter. Therefore, by obtaining a single error at the output of the network, all parameters can be changed according to the backpropagation algorithm in order to efficiently reduce the error.

In the figures which follow we apply three of the schemes we have previously seen, a linear perceptron, a non-linear perceptron with tanh activation, and a deep network, to the task of approximating a highly nonlinear process. Note that the network is configured with 4 hidden layers, a learning rate of 0.01, and is trained for 20,000 epochs. The figures depict the estimation output of the three algorithms as well as the training and test loss. The sole difference between the figures is the value of the noise variance used to generate the data. It is important to note that the main indicator of performance in the following scenarios is the test loss, rather than the training loss. This is because of the potential overfitting that may occur. Overfitting refers to the fact that the model recognises the noise as part of the signal and thus attempts to fit it. This in turn leads to very low training loss, but higher error when operating on unseen data, where the noise is completely different. Because of this effect a low training loss is not indicative of real-world performance and does not at all guarantee that the model is able to generalise well.

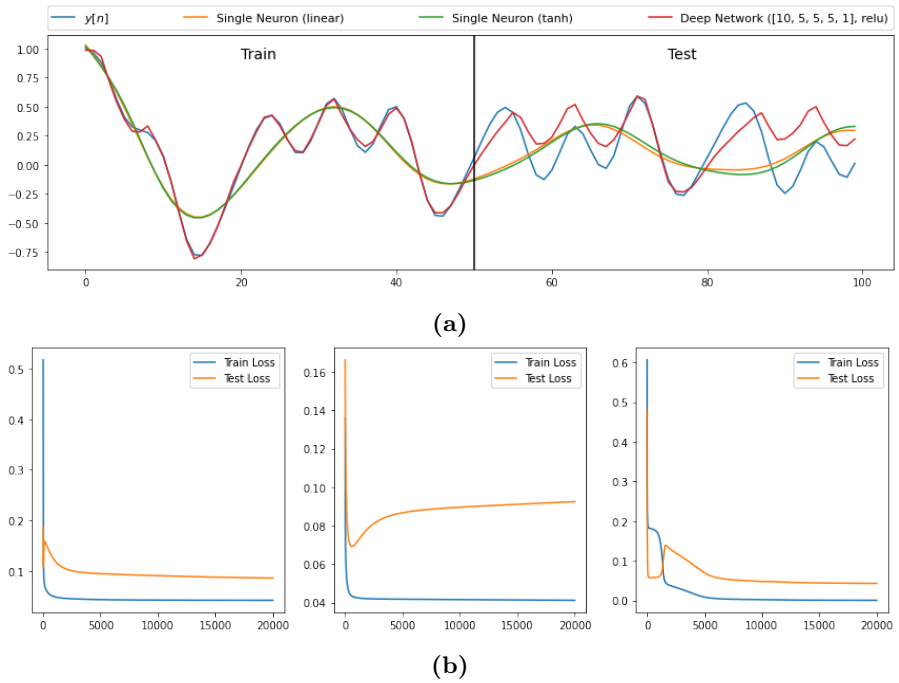
We begin with a noise variance of 0.05, as shown in Figure 42. We observe that, surprisingly, all three algorithms achieve very similar steady state loss in the test set, with both perceptron models achieving a test loss of 0.1 and the deep network just under at around 0.09. However, the linear and non-linear perceptrons converge much faster than the deep network. Finally, overfitting is not observed for any algorithm. In Figure 43 we show the same process for a noiseless input, i.e. for a noise variance of 0. In this case, the perceptron models both achieve a test loss of around 0.09, and are outperformed by the deep network which reaches a steady state test loss of 0.05.



(a)

(b)

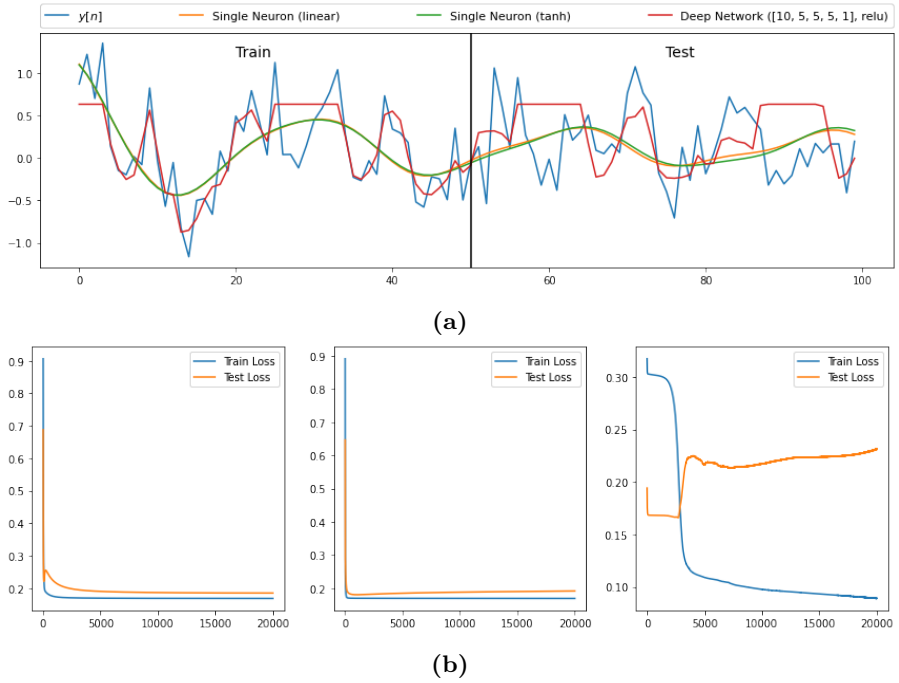
Figure 42: (a) True output and prediction outputs of linear perceptron, non-linear perceptron, and deep network and (b) corresponding learning curves of the three models (bottom)



(a)

(b)

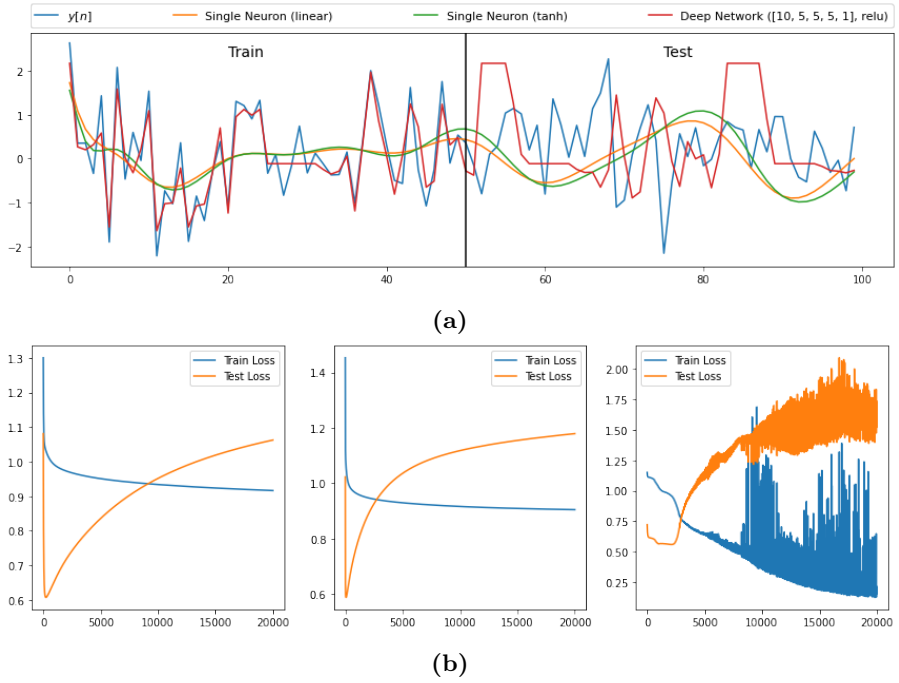
Figure 43: (a) True output and prediction outputs of linear perceptron, non-linear perceptron, and deep network and (b) corresponding learning curves of the three models (bottom)



(a)

(b)

Figure 44: (a) True output and prediction outputs of linear perceptron, non-linear perceptron, and deep network and (b) corresponding learning curves of the three models (bottom)



(a)

(b)

Figure 45: (a) True output and prediction outputs of linear perceptron, non-linear perceptron, and deep network and (b) corresponding learning curves of the three models (bottom)

In Figure 44 we test the performance of the algorithms given an input with noise variance of 0.1. In this environment of increased noise the perceptron models achieve better results, with a test loss of just under 0.2 and without any signs of overfitting. The deep network on the other hand starts to overfit at around 2500 epochs as the test loss starts increasing. That said, by implementing a technique such as early stopping, which stops training after a set amount of epochs where the test loss is monotonically increasing, the deep network could be able to provide better performance, as the minimum test loss is around 0.17. Finally, we try a very high noise variance of 1 in Figure 45. As we can see from the loss curves, in this case all three algorithms overfit to the noise in this case, and are unable to model the underlying data correctly.

Overall, we have observed that, as expected, it cannot be concluded that the deep network universally performs better than the linear or non-linear perceptron models. The main drawback, besides the much larger computational complexity, is the larger risk of overfitting to the data, especially in the presence of higher levels of noise. This stems from the model choice itself, which has enough complexity, and thus enough degrees of freedom, to attempt to fit a complicated noise signal, and is thus more prone to overfitting. In other words, in the world of signal processing and function approximators, more is not necessarily better!