

# Adaptive Signal Processing and Machine Intelligence Coursework

Prof. Danilo P. Mandic

GTAs: Edoardo Occhipinti, Mike Thornton, Sadia Sharmin, Yuyang Miao,  
Nina Moutonnet, Qiyu Rao (Kianna), Yutong Zheng

March 15, 2024

# Contents

<b>Guidelines</b>	<b>3</b>
<b>1 Classical and Modern Spectrum Estimation</b>	<b>4</b>
1.1 Properties of Power Spectral Density (PSD)	5
1.2 Periodogram-based Methods Applied to Real-World Data	5
1.3 Correlation Estimation	6
1.4 Spectrum of Autoregressive Processes	8
1.5 Real World Signals: Respiratory Sinus Arrhythmia from RR-Intervals	8
1.6 Robust Regression	9
<b>2 Adaptive signal processing</b>	<b>11</b>
2.1 The Least Mean Square (LMS) Algorithm	11
2.2 Adaptive Step Sizes	13
2.3 Adaptive Noise Cancellation	14
<b>3 Widely Linear Filtering and Adaptive Spectrum Estimation</b>	<b>16</b>
3.1 Complex LMS and Widely Linear Modelling	16
3.2 Adaptive AR Model Based Time-Frequency Estimation	17
3.3 A Real Time Spectrum Analyser Using Least Mean Square	17
<b>4 From LMS to Deep Learning</b>	<b>19</b>
4.1 Neural Networks for Prediction	19
4.2 Interpretable NNS: Convolutional Neural Network for Frequency Decomposition	21
4.2.1 Sinusoidal Signal Generation	21
4.2.2 Sinewave Frequency Classification Analysis	21
4.2.3 Kernel Initialization Impact on Convergence	24
<b>5 Tensor Decompositions for Big Data Applications</b>	<b>25</b>

## Guidelines

The coursework comprises four assignments, whose individual scores yield 80% of the final mark. The remaining 20% accounts for presentation and organisation. Students are allowed to discuss the coursework but must code their own MATLAB scripts, produce their own figures and tables, and provide their own discussion of the coursework assignments.

### General directions and notation:

- The simulations should be coded in MATLAB, a *de facto* standard in the implementation and validation of signal processing algorithms.
- The report should be clear, well-presented, and should include the answers to the assignments in a chronological order and with appropriate labelling. Students are encouraged to submit through Blackboard (**in PDF format only**), although a hardcopy submission at the undergraduate office will also be accepted.
- The report should document the results and the analysis in the assignments, in the form of figures (plots), tables, and equations, and **not** by listing MATLAB code as a proof of correct implementation.
- The students should use the following notation: boldface lowercase letters (e.g.  $\mathbf{x}$ ) for vectors, lowercase letters with a (time) argument ( $x(n)$ ) for scalar realisations of random variables and for elements of a vector, and uppercase letters ( $X$ ) for random variables. Column vectors will be assumed unless otherwise stated, that is,  $\mathbf{x} \in \mathbf{R}^{N \times 1}$ .
- In this Coursework, the typewriter font, e.g. `mean`, is used for MATLAB functions.

### Presentation:

- The length limit for the report is 42 pages. This corresponds to ten pages per assignment in addition to one page for front cover and one page for the table of contents, however, there are no page restrictions per assignment but only for the full-report (42 pages).
- The final mark also considers the presentation of the report, this includes: legible and correct figures, tables, and captions, appropriate titles, table of contents, and front cover with student information.
- The figures and code snippets (only if necessary) included in the report must be carefully chosen, for clarity and to meet the page limit.
- Do not insert unnecessary MATLAB code or the statements of the assignment questions in the report.
- For figures, (i) decide which type of plot is the most appropriate for each signal (e.g. solid line, non-connected points, stems), (ii) export figures in a correct format: without grey borders and with legible captions and lines, and (iii) avoid the use of screenshots when providing plots and data, use figures and tables instead.
- Avoid terms like *good estimate*, *is (very) close*, *somewhat similar*, etc - use formal language and quantify your statements (e.g. in dB, seconds, samples, etc).
- Note that you should submit two files to Blackboard: the report in PDF format and all the MATLAB code files compressed in a ZIP/RAR format. Name the MATLAB script files according to the part they correspond to (e.g. SEASP\_Part\_X-Y-Z.m).

### Honour code:

Students are strictly required to adhere to the College policies on students responsibilities. The College has zero tolerance to plagiarism. Any suspected plagiarism or cheating (prohibited collaboration on the coursework, including posting your solutions on internet domains and similar) will lead to a formal academic dishonesty investigation. Being found responsible for an academic dishonesty violation results in a discipline file for the student and penalties, ranging from severe reduction in marks to expulsion from College.

# 1 Classical and Modern Spectrum Estimation

**Aims:** Students will learn to:

- Understand the challenges in spectrum estimation of real-valued data.
- Consider spectral estimation as a dimensionality reduction problem and learn how to benefit from a Machine Intelligence approach to spectral estimation.
- Perform practical spectrum estimation using parametric models, understand the issues with the resolution, bias and variance, so as to motivate modern subspace approaches.
- Understand the pivotal role of correct estimates of second order statistics in applications of spectral estimations, and the effects that biased vs. unbiased estimates of the correlation function have on practical estimates of power spectra.
- Understand the benefits and drawbacks of model-based parametric and line spectra. Learn how these spectra mitigate the problems with the bias and variance of classic spectral estimators.
- Use dimensionality reduction techniques to resolve the time-frequency uncertainty issues in frequency-based Machine Intelligence.
- Learn how to deal with the problems of ill-conditioning of data correlation matrices and perform advanced Principal Component Regression type of estimation.
- Verify these concepts on real world examples in Brain Computer Interface, and in the conditioning and estimation parameters of your own Electrocardiogram (ECG) and respiration signals.
- Understand how the above concepts can be utilised in the future eHealth.

**Background.** For a discrete time **deterministic** sequence  $\{x(n)\}$ , with finite energy  $\sum_{n=-\infty}^{\infty} |x(n)|^2 < \infty$ , the Discrete Time Fourier Transform (DTFT) is defined as

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \quad (\text{DTFT}). \quad (1)$$

We often use the symbol  $X(\omega)$  to replace the more cumbersome  $X(e^{j\omega})$ . The corresponding inverse DTFT is given by

$$x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega)e^{j\omega n} d\omega \quad (\text{inverse DTFT}). \quad (2)$$

This can be verified by substituting (2) into (1). The **energy spectral density** is then defined as

$$S(\omega) = |X(\omega)|^2 \quad (\text{Energy Spectral Density}). \quad (3)$$

A straightforward calculation gives

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} S(\omega) d\omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} x(n)x(m)e^{-j\omega(n-m)} d\omega = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} x(n)x(m) \left[ \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-j\omega(n-m)} d\omega \right] = \sum_{n=-\infty}^{\infty} |x(n)|^2. \quad (4)$$

In the process, we have used the equality  $\int_{-\pi}^{\pi} e^{j\omega(n-m)} d\omega = \delta_{n,m}$  (the Kronecker delta). Equation (4) can be now restated as

$$\sum_{n=-\infty}^{\infty} |x(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} S(\omega) d\omega \quad (\text{Parseval's theorem}). \quad (5)$$

For **random sequences** we cannot guarantee finite energy for every realisation (and hence no DTFT). However, a random signal usually has a finite average power, and can therefore be characterised by average power spectral density (PSD). We assume zero mean data,  $E\{x(n)\} = 0$ , so that the autocovariance function (ACF) of a random signal  $x(n)$  is defined as

$$r(k) = E\{x(k)x^*(k-m)\} \quad (\text{Autocovariance function ACF}). \quad (6)$$

The **Power Spectral Density (PSD)** is defined as the DTFT of the ACF in the following way

$$P(\omega) = \sum_{k=-\infty}^{\infty} r(k)e^{-j\omega k} \quad \text{Definition 1 of Power Spectral Density}. \quad (7)$$

The inverse DTFT of  $P(\omega)$  is given by  $r(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} P(\omega)e^{j\omega k} d\omega$ , and it is readily verified that  $\frac{1}{2\pi} \int_{-\pi}^{\pi} P(\omega)e^{j\omega k} d\omega = \sum_{l=-\infty}^{\infty} r(l) \left[ \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega(k-l)} d\omega \right] = r(k)$ .

Observe that

$$r(0) = \frac{1}{2\pi} \int_{-\pi}^{\pi} P(\omega) d\omega. \quad (8)$$

Since from (6)  $r(0) = E\{|x(n)|^2\}$  measures the (average) signal power, the name PSD for  $P(\omega)$  is fully justified, as from (8) it represents the distribution of the (average) signal power over frequencies. The second definition of PSD is given by

$$P(\omega) = \lim_{N \rightarrow \infty} E \left\{ \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-jn\omega} \right|^2 \right\} \quad \text{Definition 2 of Power Spectral Density.} \quad (9)$$

## 1.1 Properties of Power Spectral Density (PSD)

### Approximation in the definition of PSD.

Show analytically and through simulations that the definition of PSD in (7) is equivalent to that in (9) under a mild assumption that the covariance sequence  $r(k)$  decays rapidly, that is [1] [5]

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{k=-(N-1)}^{N-1} |k| |r(k)| = 0. \quad (10)$$

Provide a simulation for the case when this equivalence does not hold. Explain the reasons.

## 1.2 Periodogram-based Methods Applied to Real-World Data

Now consider two real-world datasets: a) The sunspot time series<sup>1</sup> and b) an electroencephalogram (EEG) experiment.

- a) Apply one periodogram-based spectral estimation technique (possibly after some preprocessing) to the sunspot time series. Explain what aspect of the spectral estimate changes when the mean and trend from the data are removed (use the MATLAB commands `mean` and `detrend`). Explain how the perception of the periodicities in the data changes when the data is transformed by first applying the logarithm to each data sample and then subtracting the sample mean from this logarithmic data. [10]

### The basis for brain computer interface (BCI).

- b) The electroencephalogram (EEG) signal was recorded from an electrode located at the posterior/occipital (POz) region of the head. The subject observed a flashing visual stimulus (flashing at a fixed rate of  $X$  Hz, where  $X$  is some integer value in the range  $[11, \dots, 20]$ ). This induced a response in the EEG, known as the steady state visual evoked potential (SSVEP), at the same frequency. Spectral analysis is required to determine the value of ' $X$ '. The recording is contained in the `EEG_Data_Assignment1.mat` file<sup>2</sup> which contains the following elements: [10]
- `POz` – Vector containing the EEG samples (expressed in Volts) obtained from the POz location on the scalp,
  - `fs` – Scalar denoting the sampling frequency (1200 Hz in this case).

Read the `readme_Assignment1.txt` file for more information.

Apply the standard periodogram approach to the entire recording, as well as the averaged periodogram with different window lengths (10 s, 5 s, 1 s) to the EEG data. Can you identify the peaks in the spectrum corresponding to SSVEP? There should be a peak at the same frequency as the frequency of the flashing stimulus (integer  $X$  in the range  $[11, \dots, 20]$ ), known as the fundamental frequency response peak, and at some integer multiples of this value, known as the *harmonics* of the response. It is important to note that the subject was tired during the recording which induced a strong response within 8-10 Hz (so called alpha-rhythm), **this is not the SSVEP**. Also note that a power-line interference was induced in the recording apparatus at 50 Hz, **and this too is not the SSVEP**. To enable a fair comparison across all spectral analysis approaches, you should keep the number of frequency bins the same.

*Hint: It is recommended to have 5 DFT samples per Hz.*

How does the standard periodogram approach compare with the averaged periodogram of window length 10 s?

*Hint: Observe how straightforward it is to distinguish the estimated SSVEP peaks from other spurious EEG activity in the surrounding spectrum.*

In the case of averaged periodogram, what is the effect of making the window size very small, e.g. 1 s?

<sup>1</sup>Included in MATLAB, use `load sunspot.dat`

<sup>2</sup>Download the EEG recording from [http://www.commsp.ee.ic.ac.uk/~mandic/EEG\\_Data.zip](http://www.commsp.ee.ic.ac.uk/~mandic/EEG_Data.zip)

### 1.3 Correlation Estimation

**Unbiased correlation estimation and preservation of non-negative spectra.** Recall that the correlation-based definition of the PSD leads to the so-called *correlogram spectral estimator* given by

$$P(\omega) = \sum_{k=-(N-1)}^{N-1} \hat{r}(k) e^{j\omega k} \quad (11)$$

where the estimated autocorrelation function  $\hat{r}(k)$  can be computed using the **biased** or **unbiased** estimators given by

$$\textbf{Biased: } \hat{r}(k) = \frac{1}{N} \sum_{n=k+1}^N x(n)x^*(n-k) \quad (12)$$

$$\textbf{Unbiased: } \hat{r}(k) = \frac{1}{N-k} \sum_{n=k+1}^N x(n)x^*(n-k) \quad (0 \leq k \leq N-1). \quad (13)$$

Although it may seem that the unbiased estimate is more appropriate as its mean matches the true mean of PSD, observe that this estimate (despite being exact) can be highly erratic for larger lags  $k$  (close to  $N$ ), where fewer samples are available to estimate the PSD. As a consequence, the ACF may not be positive definite, resulting in negative PSD values.

- a) Write a MATLAB script which calculates both biased and unbiased ACF estimates of a signal and then use these ACF estimates to compute the corresponding correlogram in Eq. (11). Validate your code for different signals e.g. WGN, noisy sinusoidal signals and filtered WGN. Explain how the spectral estimates based on (12)-(13) differ from one another? In particular, how does the correlogram corresponding to the unbiased ACF estimates behave for large lags (i.e.  $k$  close to  $N$ )? Does the unbiased ACF estimate result in negative values for the estimated PSD? [10]

**Plotting the PSD in dB.** Depending on the estimation approach, the spectral estimate  $\hat{P}(\omega)$  can be asymptotically unbiased with variance  $\mu P^2(\omega)$ , where  $\mu > 0$  is a constant. When several realisations of a random signal are available, it is possible to present the estimate PSD as a confidence interval defined by  $\hat{P}(\omega) \pm \mu \hat{\sigma}_{P(\omega)}$ , where  $\hat{P}(\omega)$  and  $\hat{\sigma}_{P(\omega)}$  are respectively the mean and standard deviation of the estimated PSDs of the available observations. A drawback of this approach is that, as stated earlier, the standard deviation is proportional to the value of the PSD and therefore the confidence interval widens in zones where the PSD increases, and it is these parts that we are particularly interested in. Fig. 1 shows an overlay plot of 100 realisations of the PSD of two sinusoids immersed in i.i.d. WGN showing the mean (top), and the standard deviation of the set (bottom).

For ease of presentation, by plotting the PSD estimates in decibels we observe a more condensed realisation due to the contraction property of the logarithm.

- b) Use your code from the previous section (only the **biased** ACF estimator) to generate the PSD estimate of several realisations of a random process and plot them as in Fig. 1. Generate different signals composed of sinusoids corrupted by noise and elaborate on how disperse are the different realisation of the spectral estimate. Hint: use the `fft` and `fftshift` commands in MATLAB. [5]
- c) Plot your estimates in dB, together with their associated standard deviation (again as in Fig. 1 for comparison). How much spread out are the estimates now? Comment on the benefits of this representation. [5]

**Frequency estimation by MUSIC.** In order to accurately estimate the spectrum of closely-spaced sine waves using the periodogram, a large number of samples  $N$  is required since the frequency resolution of the periodogram is proportionate to  $1/N$ . On the other hand, subspace methods assume a harmonic model consisting of a sum of sine waves, possibly complex, in additive noise. In this setting, the noise is also complex-valued.

For illustration, consider a complex-valued signal of 30 samples in length, generated using the following code:

```
n = 0:30;
noise = 0.2/sqrt(2)*(randn(size(n))+1j*randn(size(n)));
x = exp(1j*2*pi*0.3*n)+exp(1j*2*pi*0.32*n)+ noise;
```

The signal consists of two complex exponentials (sine waves) with frequencies of 0.3 Hz and 0.32 Hz and additive complex white Gaussian noise. The noise has zero mean and variance of 0.2.

The spectral estimate using the periodogram (rectangular window, 128 frequency bins and unit sampling rate) is shown in Fig. 2. Observe that the periodogram was not able to identify the two lines in the spectrum; this is due to the resolution of the periodogram being proportionate to  $1/N$ , which is greater than the separation between the two frequencies.

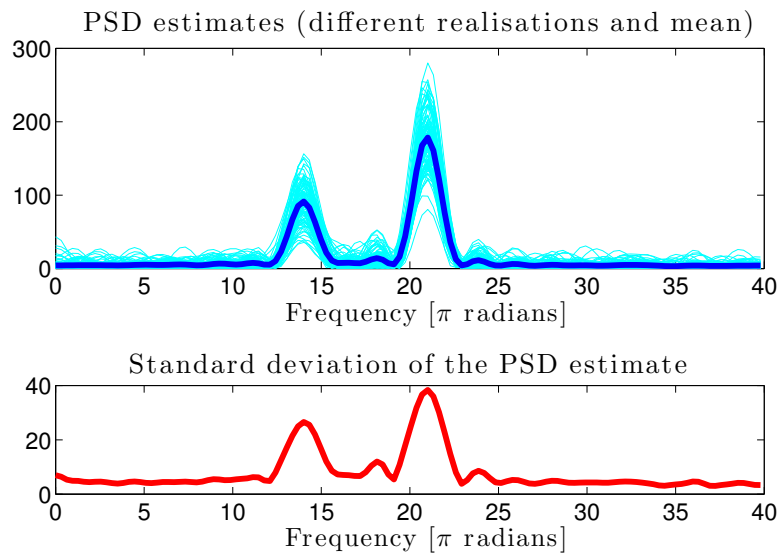


Figure 1: PSD estimates of two sinusoids immersed in noise. Top: An overlay plot of 100 realisations and their mean. Bottom: Standard deviation of the 100 estimates.

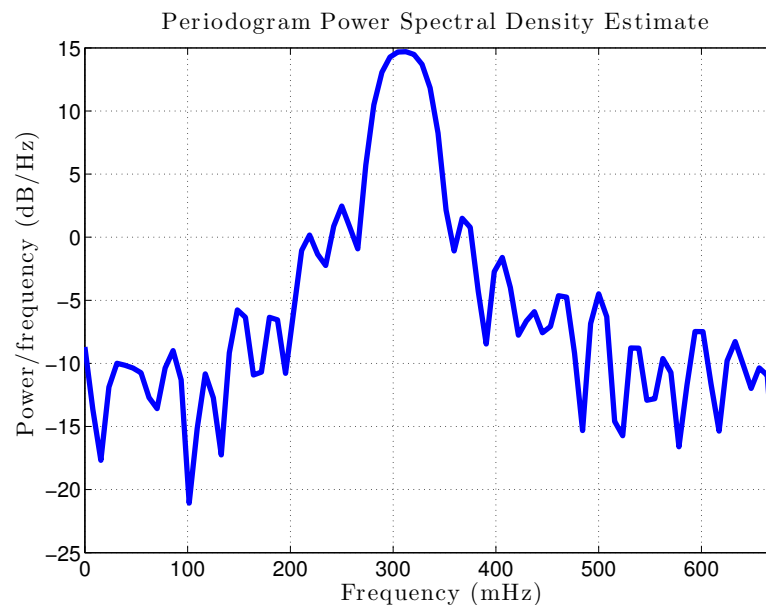


Figure 2: Periodogram of two complex exponentials with closely-spaced frequencies.

- d) Familiarise yourself with the generation of complex exponential signals, and generate signals of different frequencies and length. Verify that by considering more data samples the periodogram starts showing the correct line spectra. [5]
- e) Use the following code to find the desired line spectra using the MUSIC method.

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

Explain the operation of the first three lines in the code using the MATLAB documentation and the lecture notes. [10]  
 What is the meaning of the input arguments for the functions `corrmatrix` and `pmusic`? Does the spectrum estimated using the MUSIC algorithm provide more detailed information? State briefly the advantages and disadvantages of the periodogram and the MUSIC algorithms and comment on the bias and variance. How accurate would a general spectrum estimate be when using MUSIC?

## 1.4 Spectrum of Autoregressive Processes

In many spectrum estimation applications, only short data lengths are available; thus, classical spectrum estimation techniques based on the Fourier transform will not be able to resolve frequency elements spaced close to one another. In order to solve this problem, we can use modern spectrum estimation methods based on the pole-zero modelling of the data.

Consider a general ARMA( $p, q$ ) process given by

$$y(n) = a_1 y(n-1) + \dots + a_p y(n-p) + w(n) + b_1 w(n-1) + \dots + b_q w(n-q)$$

The power spectrum of  $y$  has the form

$$P_y(e^{j\omega}) = \frac{|\sum_{k=1}^q b_k e^{-jk\omega}|^2}{|1 - \sum_{k=1}^p a_k e^{-jk\omega}|^2}$$

Thus, the power spectrum can be estimated through the parameters  $(a_1, \dots, a_p, b_1, \dots, b_q)$ . **The assumption of an underlying model for the data is the key difference between classical and modern spectrum estimation methods.**

For an AR process in particular, the power spectrum is the output of an all-pole filter given by

$$P_y(e^{j\omega}) = \frac{\sigma_w^2}{|1 - \sum_{k=1}^p a_k e^{-jk\omega}|^2}$$

The parameters  $\sigma_w^2$  and  $\mathbf{a} = [a_1 \dots a_p]^T$  can be estimated by a set of  $(p+1)$  linear equations

$$\begin{bmatrix} r_x(0) & r_x(1) & \cdot & \cdot & \cdot & r_x(p) \\ r_x(1) & r_x(0) & \cdot & \cdot & \cdot & r_x(p-1) \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ r_x(p) & r_x(p-1) & \cdot & \cdot & \cdot & r_x(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_p \end{bmatrix} = \sigma_w^2 \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

where  $r_x(k)$  could be calculated using the biased autocorrelation estimate

$$r_x(k) = \frac{1}{N} \sum_{n=0}^{N-1-k} x(n+k)x(n)$$

or the unbiased autocorrelation estimate

$$r_x(k) = \frac{1}{N-k} \sum_{n=0}^{N-1-k} x(n+k)x(n)$$

- Based on your answers in Section 2.1, elaborate on the shortcomings of using the unbiased ACF estimate when finding the AR parameters? [see Eq. (13)] [5]
- Generate 1000 samples of data in MATLAB, according to the following equation [10]

$$x(n) = 2.76x(n-1) - 3.81x(n-2) + 2.65x(n-3) - 0.92x(n-4) + w(n)$$

where  $w \sim \mathcal{N}(0, 1)$  and discard the first 500 samples (`x=x(500:end)`) to remove the transient output of the filter. Estimate the power spectrum density of the signal using model orders  $p = 2, \dots, 14$  and comment on the effects of increasing the order of the (assumed) underlying model by comparing the estimation to the true Power Spectral Density. Only plot the results of the model orders which produced the best results.

- Repeat the experiment in b) for data length of 10,000 samples. What happens to the PSD when the chosen model order is lower (under-modelling) or higher (over-modelling) than the correct AR(4) model order? [5]

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

**Important change Section 1.5 of the CW:**

- If you have taken Statistical Signal Processing & Inference last year, then you already have your own ECG data from the wrists, and please proceed with this Assignment;**
- If you do not have your own ECG recordings, then we will provide the data. We will an email to the class with the data and explanations.**



Respiratory sinus arrhythmia (RSA) refers to the modulation of cardiac function by respiratory effort. This can be readily observed by the speeding up of heart rate during inspiration (“breathing in”) and the slowing down of heart rate during expiration (“breathing out”). The strength of RSA in an individual can be used to assess cardiovascular health. Breathing at regular rates will highlight the presence of RSA in the cardiac (ECG) data.

- Apply the standard periodogram as well as the averaged periodogram with different window lengths (e.g. 50 s, 150 s) to obtain the power spectral density of the RRI data. Plot the PSDs of the RRI data obtained from the three trials separately. [10]
- Explain the differences between the PSD estimates of the RRI data from the three trials? Can you identify the peaks in the spectrum corresponding to frequencies of respiration for the three experiments? [5]
- Plot the AR spectrum estimate for the RRI signals for the three trials<sup>3</sup>. To find the optimal AR model order, experiment with your model order until you observe a peak in the spectrum (approximately) corresponding to the theoretical respiration rate. List the differences you observe between your estimated AR spectrum and the periodogram estimate in Part a). [10]

## 1.6 Robust Regression

Load the file<sup>4</sup> `PCAPCR.mat` which includes the data matrices  $\mathbf{X} \in \mathbb{R}^{N \times d_x}$  and  $\mathbf{Y} \in \mathbb{R}^{N \times d_y}$ , described below.

Training Data	Testing Data	Note
$\mathbf{X}$	-	Input variables, some of which are collinear. Each column represents $N$ measurements of an input variable.
$\mathbf{X}_{\text{noise}}$	$\mathbf{X}_{\text{test}}$	Noise corrupted input matrix $\mathbf{X}_{\text{noise}} = \mathbf{X} + \mathbf{N}_{\mathbf{X}}$ , where the elements of $\mathbf{N}_{\mathbf{X}}$ were drawn from a zero-mean Gaussian distribution.
$\mathbf{Y}$	$\mathbf{Y}_{\text{test}}$	Output variables obtained from $\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{N}_{\mathbf{Y}}$ where the coefficient matrix $\mathbf{B}$ is unknown. Each column in $\mathbf{Y}$ represents $N$ measurements of an output variable. The elements of $\mathbf{N}_{\mathbf{Y}}$ were drawn from a zero-mean Gaussian distribution.

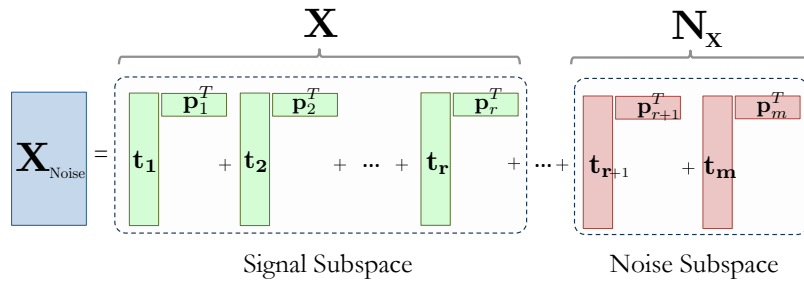


Figure 3: Principle of PCA: Illustration of the signal and noise subspaces.

Using the Matlab command `svd`, obtain the singular value decomposition for the matrices  $\mathbf{X}$  and  $\mathbf{X}_{\text{noise}}$ .

- Plot the singular values of  $\mathbf{X}$  and  $\mathbf{X}_{\text{noise}}$  (hint: use the `stem` command), and identify the rank of the input data  $\mathbf{X}$ . Plot the square error between each singular value of  $\mathbf{X}$  and  $\mathbf{X}_{\text{noise}}$ . Explain the effect of noise on the singular values, and state at what point would it become hard to identify the rank of the matrix  $\mathbf{X}_{\text{noise}}$ . [5]
- Using only the  $r$  most significant principal components (as determined by the identified rank), create a low-rank approximation of  $\mathbf{X}_{\text{noise}}$ , denoted by  $\tilde{\mathbf{X}}_{\text{noise}}$ . Compare the difference (error) between the variables (columns) of the noiseless input matrix,  $\mathbf{X}$ , and those in the noise corrupted matrix  $\mathbf{X}_{\text{noise}}$  and denoised matrix  $\tilde{\mathbf{X}}_{\text{noise}}$ . [5]

The output data are obtained as  $\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{N}_{\mathbf{Y}}$ . The ordinary least squares (OLS) estimate for the unknown regression matrix,  $\mathbf{B}$ , is then given

$$\hat{\mathbf{B}}_{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \quad (14)$$

Since the matrix  $\mathbf{X}^T \mathbf{X}$  which is calculated from the original data,  $\mathbf{X}$ , is sub-rank, the OLS solution in (14) becomes intractable. On the other hand, for the noisy data,  $\mathbf{X}_{\text{noise}}$ , the term  $\mathbf{X}_{\text{noise}}^T \mathbf{X}_{\text{noise}}$  is full-rank, and therefore admits the OLS solution, however, this may introduce spurious correlations in the calculation of regression coefficients.

<sup>3</sup>Use the MATLAB function `aryule` to estimate the AR coefficients for your RRI signal.

<sup>4</sup>Download the PCR files from: <http://www.commsp.ee.ic.ac.uk/~mandic/PCR.zip>

To circumvent this issue in the estimation of  $\mathbf{B}$ , the principal component regression (PCR) method first applies principal component analysis (PCA) on the input matrix  $\mathbf{X}_{\text{noise}}$ . Specifically, the SVD of  $\mathbf{X}_{\text{noise}}$  is given by  $\mathbf{X}_{\text{noise}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ . By retaining the  $r$  largest principal components ( $r$ -singular values and the associated singular vectors), the PCR solution is given by

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

where the subscript  $(1:r)$  denotes the  $r$ -largest singular values and the corresponding singular vectors. In this way, the PCR solution avoids both the problem of collinearity and noise in the input matrix. Figure 4 illustrates the difference between the OLS and PCR methods.

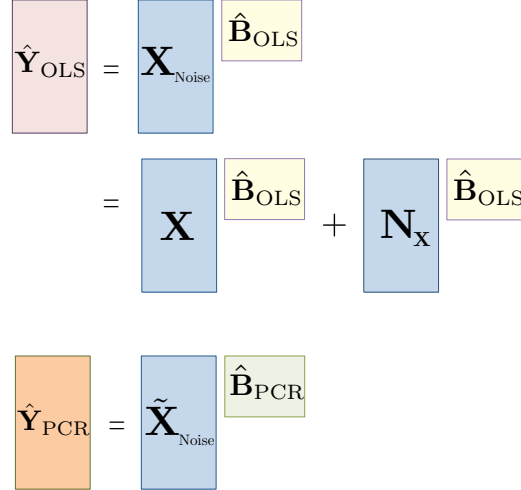


Figure 4: Comparing OLS and PCR solutions.

- c) Calculate the OLS and PCR solutions for the parameter matrix  $\mathbf{B}$ , which relates  $\mathbf{X}_{\text{noise}}$  and  $\mathbf{Y}$ . Next, compare the estimation error between  $\mathbf{Y}$  and  $\hat{\mathbf{Y}}_{\text{OLS}} = \mathbf{X}_{\text{noise}}\hat{\mathbf{B}}_{\text{OLS}}$  and  $\hat{\mathbf{Y}}_{\text{PCR}} = \tilde{\mathbf{X}}_{\text{noise}}\hat{\mathbf{B}}_{\text{PCR}}$ . Explain what happens when you estimate the data from the test-set using the regression coefficients computed from the training set, and quantify the performance by comparing  $\mathbf{Y}_{\text{test}}$  and  $\hat{\mathbf{Y}}_{\text{test-OLS}} = \mathbf{X}_{\text{test}}\hat{\mathbf{B}}_{\text{OLS}}$  with  $\hat{\mathbf{Y}}_{\text{test-PCR}} = \tilde{\mathbf{X}}_{\text{test}}\hat{\mathbf{B}}_{\text{PCR}}$ . [5]

In real world machine intelligence applications, a model is trained with a finite set of data which is referred to as the training set. After the training, the model is not only expected to be a good fit to the training data, but also it needs to model out-of-sample data. Any model which fits the training data well but has poor out-of-sample performance is said to be “over-fitted”. Therefore, it is important to validate the regression model computed in this section on a test-set which is another realisation of the signal drawn from the statistical distribution of the training set. For this task, the file `PCAPCR.mat` contains both the training data and test data, which should be used to validate the effectiveness of the regression model derived from the OLS and PCR solutions.

- d) The best way to assess the effectiveness of the PCR compared to the OLS solution is by testing the estimated regressions coefficients,  $\hat{\mathbf{B}}$ , over an ensemble of test data. The file `PCR.zip` contains the script `regval`, the output of which is a new realisation of the test data,  $\mathbf{Y}$ , and its estimate,  $\hat{\mathbf{Y}}$ , the input are the regression coefficients, and the function syntax is: [5]

$$[\hat{\mathbf{Y}}, \mathbf{Y}] = \text{regval}(\hat{\mathbf{B}}).$$

Using the *same* PCR and OLS regression coefficients as in (c), compute and compare the mean square error estimates for the PCR and OLS schemes,  $\text{MSE} = E\{\|\mathbf{Y} - \hat{\mathbf{Y}}\|_2^2\}$ , based on the realisations of  $\mathbf{Y}$  and  $\hat{\mathbf{Y}}$  provided by the function `regval`. Comment on the effectiveness of these schemes.

## 2 Adaptive signal processing

**Aims:** Students will learn to:

- Design and implement adaptive filtering algorithms for the processing of online streaming data.
- Learn how adaptive filters can be seen as implementation of time-varying autoregressive (AR) models, and understand the link with subspace methods for spectral estimation.
- Understand the components and learning strategies for adaptive filters and identify the appropriate configurations for different applications.
- Explain and demonstrate the effects of the various algorithm parameters on the performance of adaptive filters.
- Demonstrate the effects of the input data structure on the convergence of adaptive filtering algorithms.
- Learn how adaptive filters deal with nonstationary streaming data, and understand how they deal with the existence, uniqueness, and bias–variance trade–off of the solutions.
- Derive theoretical performance measures of the adaptive algorithms and compare them to experimental results.
- Derive optimisation strategies with an adaptive stepsize and show how these approach the performance of state–space based Kalman filters.
- Apply linear adaptive filters in real–world applications of adaptive noise cancellation and in Brain Computer Interface.
- Acquire hands–on experience on the utility of linear adaptive filtering architectures in Big Data scenarios, such as in the processing of online streaming data and learn how to employ the knowledge of the underlying data structure to enhance performance.

### 2.1 The Least Mean Square (LMS) Algorithm

To study nonstationary signals, adaptive filters that approximate, in a recursive fashion, the Wiener solution are considered. The simplest adaptive filter is based on the least mean square (LMS) algorithm, which recursively updates the weights using a gradient-based minimisation of the (instantaneous error) cost function  $J(n) = \frac{1}{2}e^2(n)$  according to

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n), \quad n = 0, 1, \dots \quad (15)$$

where  $\mu > 0$  is the adaptation gain which controls the stability of the algorithm, and  $\mathbf{w}(0)$  can be set to any value (usually  $\mathbf{w}(0) = \mathbf{0}$ ). Notice that the time-varying weight vector of the LMS filter  $\mathbf{w}(n)$  allows for the estimation of non-stationary signals.

The error  $e(n) = x(n) - \hat{x}(n)$  is calculated as the difference between the original process  $x(n)$  and the estimate  $\hat{x}(n)$ , given by

$$\hat{x}(n) = \mathbf{w}^T(n)\mathbf{x}(n) = \sum_{m=1}^M w_m(n)x(n-m). \quad (16)$$

**Note:**  $x(n)$ ,  $\hat{x}(n)$  and  $e(n)$  are scalar quantities, whereas  $\mathbf{w}(n)$  and  $\mathbf{x}(n)$  are  $M \times 1$  column vectors.

**Identification of AR processes.** Linear adaptive filters have a wide range of applications. Speech denoising, for instance, involves non-stationary signals, and therefore requires adaptive estimation. Consider the structure shown in Figure 5 consisting of two separate stages: the **synthesis** of a signal from white noise using an autoregressive model, and the **analysis** stage which performs the adaptation of coefficients to recreate the original signal. We now consider the problem of adaptive linear prediction using the LMS algorithm. Let  $x(n)$  be a second-order auto-regressive process satisfying the difference equation

$$x(n) = a_1x(n-1) + a_2x(n-2) + \eta(n) \quad (17)$$

where  $\eta(n) \sim \mathcal{N}(0, \sigma_\eta^2)$ . Your aim is to implement an LMS estimator of the AR coefficients  $a_1$  and  $a_2$  using the signal  $x(n)$ .

- The LMS filter can be seen as a dynamical system for which the inputs are  $x(n-1)$ ,  $x(n-2)$  and the outputs represent the estimates  $\hat{a}_1$ ,  $\hat{a}_2$  and  $\hat{x}(n)$ . If the original parameters are  $a_1 = 0.1$ ,  $a_2 = 0.8$  and  $\sigma_\eta^2 = 0.25$ , what are the entries of the correlation matrix of the input vector  $\mathbf{x}(n) = [x(n-1), x(n-2)]^T$ ? (Hint: It will be  $2 \times 2$  matrix.) Furthermore, for what range of values of the step size  $\mu$  will the LMS converge in the mean? [5]

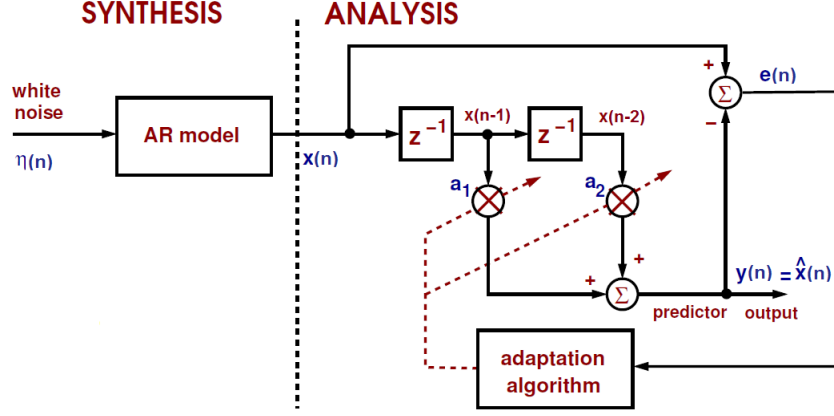


Figure 5: Synthesis and analysis structure for an AR model.

- b) Implement an LMS adaptive predictor using  $N = 1000$  samples of  $x(n)$  as in a) and plot the squared prediction error  $e^2(n)$  dB i.e.  $10 \log(e^2(n))$  along time using step sizes  $\mu = 0.05$  and  $\mu = 0.01$ . Repeat this experiment for 100 different realizations of  $x(n)$  and plot the **learning curve** by averaging the plots of  $10 \log(e^2(n))$ . [5]

The LMS weight estimates fluctuate around the optimal solution and these fluctuations introduce **excess error** to the minimum achievable error signal. The additional error power introduced by the adaptive filter can be quantified by the excess mean square error (EMSE) metric which measures 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 the ratio of the excess mean square error and the minimum mean square error

$$\text{MSE} = \lim_{n \rightarrow \infty} \mathbb{E}\{e^2(n)\} = \sigma_\eta^2 + \text{EMSE} \quad (18)$$

$$\mathcal{M} = \frac{\text{EMSE}}{\sigma_\eta^2}. \quad (19)$$

For small step-sizes, the misadjustment of the LMS is approximated as

$$\mathcal{M}_{\text{LMS}} \approx \frac{\mu}{2} \text{Tr}\{\mathbf{R}\} \quad (20)$$

where  $\mathbf{R}$  is the auto-correlation matrix of the input,  $\mu$  is the learning rate and  $\text{Tr}\{\cdot\}$  denotes the matrix trace operator.

- c) For the example in Part a), estimate the corresponding misadjustment by time-averaging over the steady state of the ensemble-averaged learning curves using 100 independent trials of the experiment. Compare the estimated values with the theoretical LMS misadjustment in (20). [5]
- d) Estimate the steady state values of the adaptive filter coefficients for the step-sizes of  $\mu = 0.05$  and  $\mu = 0.01$ . Note that you may do this by averaging the steady-state values of the coefficients (obtained along the the final iterations of the LMS) over 100 independent trials of the experiment. Compare these estimated values with the true coefficients, and explain your findings. [5]

### The Leaky LMS Algorithm

When the input signal  $\mathbf{x}(n)$  has an autocorrelation matrix with zero eigenvalues, the LMS adaptive filter has one or more weights that do not converge and this may lead to instability. One way to stabilize the algorithm is to introduce a *leakage* coefficient,  $0 < \gamma < 1$ , to force the filter weights to converge.

$$\text{Leaky LMS: } \mathbf{w}(n+1) = (1 - \mu\gamma) \mathbf{w}(n) + \mu e(n) \mathbf{x}(n) \quad (21)$$

- e) Derive the LMS coefficient update for  $\mathbf{w}(n)$  that minimizes the cost function  $J_2(n) = \frac{1}{2} (e^2(n) + \gamma \|\mathbf{w}(n)\|_2^2)$  and show that you obtain the leaky LMS equation shown in Equation (21). [5]
- f) Implement the leaky LMS algorithm (with different values for  $\gamma$  and  $\mu$ ) to estimate the AR coefficients of the signal given in Part a). Why do the weights of the leaky LMS algorithm converge to incorrect values for the parameters  $a_1$  and  $a_2$ ? [5]

## 2.2 Adaptive Step Sizes

Choosing the step-size for the LMS is a challenging task due to the trade-off between convergence speed and steady state error variance. Ideally, the step-size should be large in the beginning of the adaptation and small as the parameter estimates become closer to the true value, thus allowing for fast convergence and reduced steady-state error. To this end, variable step-size (VSS) algorithms aim to solve two simultaneous unconstrained optimisation procedures – to minimise the cost function with respect to the weight vector, while at the same time minimising the cost function with respect to the step size. Consider the LMS algorithm

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu(n)e(n)\mathbf{x}(n) \quad (22)$$

where the learning rate  $\mu(n)$  is made adaptive using gradient adaptive step-size (GASS) algorithms based on the minimisation of  $\nabla_{\mu} J$ , such as the algorithms introduced by Benveniste [2], Ang & Farhang [3], and Matthews & Xie [4]. Within GASS, the learning rate  $\mu(n)$  is updated according to

$$\mu(n+1) = \mu(n) + \rho e(n)\mathbf{x}^T(n)\psi(n) \quad (23)$$

where the term  $\psi(n)$  can take one of the following forms

$$\text{Benveniste : } \psi(n) = [\mathbf{I} - \mu(n-1)\mathbf{x}(n-1)\mathbf{x}^T(n-1)] \psi(n-1) + e(n-1)\mathbf{x}(n-1) \quad (24)$$

$$\text{Ang \& Farhang : } \psi(n) = \alpha\psi(n-1) + e(n-1)\mathbf{x}(n-1), \quad 0 < \alpha < 1 \quad (25)$$

$$\text{Matthews \& Xie : } \psi(n) = e(n-1)\mathbf{x}(n-1) \quad (26)$$

- a) Implement the three GASS algorithms above, and compare their performance when identifying a real-valued MA(1) system [15]

$$x(n) = 0.9\eta(n-1) + \eta(n) \quad \eta \sim \mathcal{N}(0, 0.5). \quad (27)$$

Explain the performance advantages/disadvantages against a standard LMS algorithm (with a fixed step size,  $\mu = 0.01$  and  $\mu = 0.1$ ). Plot the weight error curves ( $\tilde{w}(n) = w_o - w(n)$  where  $w_o = 0.9$ ) for each algorithm and comment on their convergence speed and steady state error.

Since the weight update of the LMS is proportional to the input vector  $\mathbf{x}(n)$ , it is often useful to normalise the step size by using the normalized LMS (NLMS) algorithm in the form

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\beta}{\epsilon + \|\mathbf{x}(n)\|^2} e(n)\mathbf{x}(n) = \mathbf{w}(n) + \frac{\beta}{\epsilon + \mathbf{x}^T(n)\mathbf{x}(n)} e(n)\mathbf{x}(n) \quad (28)$$

where the regularization factor  $\epsilon$  prevents the algorithm from becoming unstable if the input vector is zero.

- b) Verify that the update equation [5]

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e_p(n)\mathbf{x}(n) \quad (29)$$

based upon the *a posteriori* error  $e_p(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n+1)$  is equivalent to the NLMS algorithm. Comment upon the relationship between  $\beta$ ,  $\epsilon$  and  $\mu$ .

*Hint: Start by expressing  $\Delta\mathbf{w}(n) = \mathbf{w}(n+1) - \mathbf{w}(n)$  in terms of the *a priori* error  $e(n)$  as*

$$\Delta\mathbf{w}(n) = \mu e(n) \left[ 1 - \mu \frac{\|\mathbf{x}(n)\|^2}{1 + \mu\|\mathbf{x}(n)\|^2} \right] \mathbf{x}(n) \quad (30)$$

The gain of the NLMS algorithm can be made adaptive by making the regularization factor  $\epsilon \rightarrow \epsilon(n)$  time varying. The generalized normalized gradient descent (GNGD) algorithm [5] adapts  $\epsilon(n)$  via a gradient method, and is given by

$$\epsilon(n+1) = \epsilon(n) - \rho \mu \frac{e(n)e(n-1)\mathbf{x}^T(n)\mathbf{x}(n-1)}{(\epsilon(n-1) + \|\mathbf{x}(n-1)\|^2)^2} \quad (31)$$

- c) Implement the GNGD algorithm for the system identification problem in part (a), plot the weight estimates and compare it with Benveniste's algorithm. Compare the computational complexity (number of multiplications and additions every time instant  $n$ ) of the Benveniste's GASS algorithm to the GNGD. [10]

## 2.3 Adaptive Noise Cancellation

This exercise deals with noise suppression for two de-noising adaptive filtering configurations.

- **Adaptive Line Enhancer (ALE).** An ALE consists of a delay operator and a linear predictor, as illustrated in Fig 6. The noise-corrupted signal  $s(n)$  is composed of a clean signal  $x(n)$  and a noise term (not necessarily white),  $\eta(n)$ . The delay,  $\Delta$  must be chosen such that noise in the signal  $s(n)$  and the predictor input  $\mathbf{u}(n) = [s(n - \Delta), \dots, s(n - \Delta - M + 1)]^T$  are uncorrelated, so that only the noise (and not the signal  $x(n)$ ) is suppressed by the linear predictor. This linear predictor is an FIR filter and its weights are adjusted using an adaptive algorithm, usually the LMS algorithm.

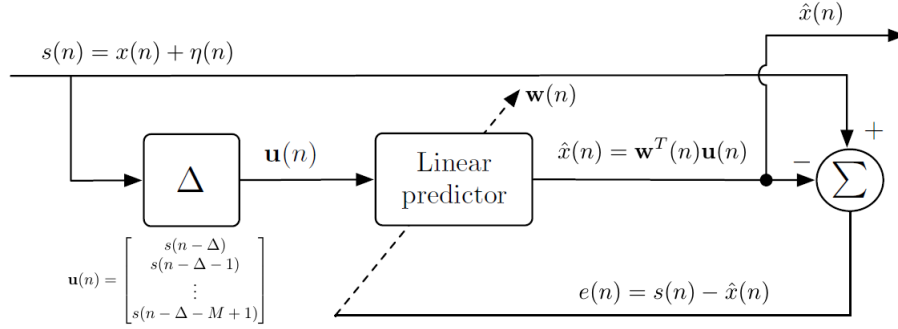


Figure 6: Adaptive line enhancement configuration.

- **Adaptive Noise Cancellation (ANC).** This configuration receives the noise-corrupted primary input  $s(n)$ , which contains the corrupted signal, and a secondary noise input  $\epsilon(n)$ , which is correlated in some unknown way with the primary noise. For example, in a mobile phone, the primary input comes from the microphone that the user talks into and the secondary input may come from an external microphone used to pick-up the ambient noise. A linear predictor is used to estimate the noise present in the signal,  $\eta(n)$ , using the secondary noise measurement  $\epsilon(n)$  and then the clean signal  $\hat{x}(n)$  is formed by subtracting  $\hat{\eta}(n)$  from  $s(n)$ . The ANC configuration is shown in Fig 7.

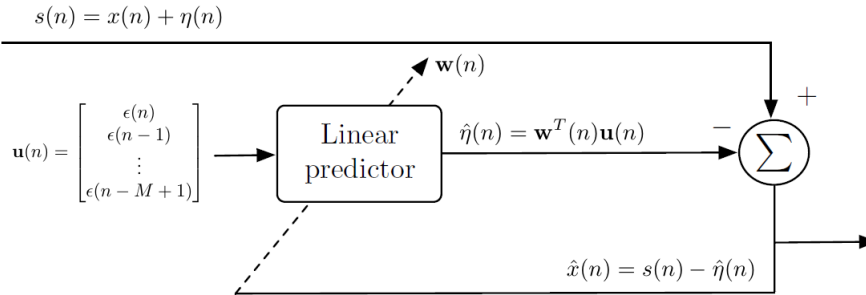


Figure 7: Adaptive noise cancellation configuration.

For both the noise suppression configurations, the prediction error  $\varepsilon(n)$  is defined as the difference between the clean signal,  $x(n)$  and the estimated de-noised signal  $\hat{x}(n)$ . The estimate of the mean square prediction error (MSPE) is

$$\text{MSPE} = \frac{1}{N} \sum_{n=0}^{N-1} \varepsilon^2(n) = \frac{1}{N} \sum_{n=0}^{N-1} (x(n) - \hat{x}(n))^2. \quad (32)$$

Generate a noise-corrupted signal

$$s(n) = x(n) + \eta(n) \quad (33)$$

where  $x(n)$  is a unit amplitude sinusoid of angular frequency  $\omega_o = 0.01\pi$  and  $\eta(n)$  is coloured noise generated using the moving average filter

$$\eta(n) = v(n) + 0.5v(n - 2) \quad (34)$$

where  $v(n)$  is white noise with unit variance.

- a) Given the model for the noise in (34), what is the minimum value for the delay  $\Delta$  that may be used in the adaptive line enhancer in Fig 6 for a filter length of  $M > 1$ ? *Hint: Start by expanding the Mean Square Error  $\mathbb{E}\{(s(n) - \hat{x}(n))^2\}$  to determine how the correlation in the noise affects the linear predictor.* [10]  
Write a MATLAB program for the adaptive line enhancement using the LMS algorithm and justify your choice of minimum  $\Delta$ .
- b) Generate 1000 samples of  $s(n)$  and use your MATLAB program to estimate  $x(n)$  for filter orders  $M = 5, 10, 15, 20$ . [10]  
Use values for  $\Delta$  that range from the minimum value determined in part (a) to  $\Delta = 25$ . What is the dependence (if any) between the delay  $\Delta$  and the MSPE (i.e. how well the de-noising algorithm has performed)?  
What is the effect of the filter order  $M$  on the MSPE? Given that the computational cost increases with  $M$ , what filter order would be a good pragmatic choice?
- c) Write a MATLAB program for the adaptive noise cancellation configuration (see Fig 7). Use the MSPE measure in (32) to compare the relative performance of the ANC and ALE in de-noising the noisy sinusoid in (33). [5]
- d) Load a single-channel EEG Data<sup>5</sup> (either Cz or POz) from the file `EEG_Data_Assignment2.mat`. As seen in the previous assignments, there is a strong 50 Hz component introduced by the mains. Using the ANC configuration, remove the 50 Hz component by generating a synthetic reference input composed of a sinusoid of 50 Hz corrupted by white Gaussian noise. (*Hint: Experiment with different step-sizes,  $\mu$  and filter lengths,  $M$ , to suppress the 50 Hz component without affecting the other frequency components*). Plot the spectrum for the corrupted and denoised EEG data using the `spectrogram` function with appropriate window length and overlap. What do you observe? [15]

---

<sup>5</sup>Download the EEG data from [http://www.commsp.ee.ic.ac.uk/~mandic/EEG\\_Data.zip](http://www.commsp.ee.ic.ac.uk/~mandic/EEG_Data.zip)

### 3 Widely Linear Filtering and Adaptive Spectrum Estimation

**Aims:** Students will learn to:

- Understand the concept of optimal directions of complex gradient, the  $\mathbb{C}\mathbb{R}$  calculus, and the utility of the pseudo-statistics for noncircular complex data (pseudo-covariance).
- Familiarise themselves with the concepts that underpin the advantages of complex-valued adaptive learning systems when processing bivariate data, such as the use of the phase information and the compact forms of correlation and augmented correlation matrices.
- Implement widely linear adaptive filtering configurations for the estimation of noncircular signals, derive the Augmented Complex Least Mean Square (ACLMS), and understand its potential in streaming Big Data scenarios.
- Use widely linear modelling in real-time scenarios of renewable energy and smart grid applications, such as 2D wind prediction and frequency estimation in Smart Grid.
- Use adaptive filters for spectral estimation of non-stationary data, and use time-frequency representations to demonstrate the advantages of this online implementations over classical subspace techniques in Assignment 1.
- Demonstrate the benefits of complex-valued adaptive systems on an example of the nonstationary and multichannel Electroencephalogram (EEG) estimation in Brain Computer Interface.

#### 3.1 Complex LMS and Widely Linear Modelling

Complex-valued signals play a pivotal role in communications, array signal processing, biomedical signal processing and related fields. These signals can be either complex by design, such as symbols used in data communication, or are made complex by convenience of representation e.g. 2D wind speed and direction. The complex LMS (CLMS) is one of the most widely used estimation algorithms to process complex valued data and is given by

$$\begin{aligned} y(n) &= \mathbf{h}^H(n)\mathbf{x}(n) \\ e(n) &= y(n) - \hat{y}(n) \\ \mathbf{h}(n+1) &= \mathbf{h}(n) + \mu e^*(n)\mathbf{x}(n) \end{aligned} \quad (35)$$

where  $\mu$  is the learning rate. The CLMS is similar, in form, to its real-valued counterpart, (15), used in Part 3.

Although statistical tools for complex random variables are traditionally treated as generic extensions of their real-valued counterparts, this approach is valid only for circular random variables. It has now been accepted that the standard strictly linear model for complex data is not guaranteed to capture the complete second-order statistical relationship between the input and the output as generic strictly linear extensions of real-valued estimators cater only for circular data i.e. data with rotation invariant probability distributions.

To account for general complex data, we have to employ the widely linear framework. By using an augmented input vector,  $\mathbf{x}^a = [\mathbf{x}^T \ \mathbf{x}^H]^T$ , that contains the input vector and its conjugate, the widely linear model is equipped with the sufficient degrees of freedom to fully exploit the second order statistics in the data. The Augmented CLMS (ACLMS) [6] is given by [7]

$$\begin{aligned} \hat{y}(n) &= \mathbf{h}^H(n)\mathbf{x}(n) + \mathbf{g}^H(n)\mathbf{x}^*(n) \\ e(n) &= y(n) - \hat{y}(n) \\ \mathbf{h}(n+1) &= \mathbf{h}(n) + \mu e^*(n)\mathbf{x}(n) \\ \mathbf{g}(n+1) &= \mathbf{g}(n) + \mu e^*(n)\mathbf{x}^*(n). \end{aligned} \quad (36)$$

- a) Generate a first-order widely-linear-moving-average process, WLMA(1), driven by circular white Gaussian noise,  $x(n)$  [10]

$$y(n) = x(n) + b_1 x(n-1) + b_2 x^*(n-1) \quad x \sim \mathcal{N}(0, 1) \quad (37)$$

where  $b_1 = 1.5 + 1j$  and  $b_2 = 2.5 - 0.5j$ . Write a MATLAB function for the ACLMS and implement both the CLMS and ACLMS in the system identification setting to identify the WLMA model in (37). Plot the learning curve,  $10\log|e(n)|^2$ , for the ACLMS and CLMS. Comment on the steady state error of the ACLMS and CLMS.

*Hint: To obtain a smoother learning curve, plot the ensemble average of the learning curve from 100 independent simulations.*

- b) Load the bivariate wind data<sup>6</sup> of the wind speeds in the East-West direction,  $v_{\text{east}}$ , and North-South direction,  $v_{\text{north}}$ . [10]  
Form a complex-valued wind signal

$$v[n] = v_{\text{east}}[n] + jv_{\text{north}}[n] \quad (38)$$

for the three wind regimes (low, medium, high). Use the `scatter(x, y)` function to plot a scatter diagram (the scatter diagram of the real and imaginary parts of a signal is also referred to as a circularity plot) for these three regimes. Comment on the circularity of the complex wind signal for low, medium and high wind speeds. Configure the CLMS and ACLMS filters in a prediction setting (see Equation (16)) to perform a one-step ahead prediction of the complex wind data. Experiment with different filter lengths and comment on which algorithm (CLMS or ACLMS) performs better for the different wind regimes.

<sup>6</sup>Download the wind data from <http://www.commsp.ee.ic.ac.uk/~mandic/wind-dataset.zip>



### 3.2 Adaptive AR Model Based Time-Frequency Estimation

In Part 2.2, the AR spectrum was estimated by assuming stationary signals. Therefore, the block-based estimate of the AR coefficients (and the corresponding power spectrum) will be inaccurate when the signal is non-stationary. In this section, the concepts of non-stationary (online) spectrum estimation will be explored.

- a) Generate the frequency modulated (FM) signal  $y(n) = e^{j(\frac{2\pi}{f_s}\phi(n))} + \eta(n)$  where  $\eta(n)$  is circular complex-valued white noise with zero mean and variance  $\sigma_\eta^2 = 0.05$  and the phase  $\phi(n) = \int f(n) dn$  is generated as<sup>7</sup> [10]

$$f(n) = \frac{d\phi(n)}{dn} = \begin{cases} 100, & 1 \leq n \leq 500 \\ 100 + \frac{n-500}{2}, & 501 \leq n \leq 1000 \\ 100 + \left(\frac{n-1000}{25}\right)^2, & 1001 \leq n \leq 1500 \end{cases} \quad (39)$$

Use the `aryule` function find the AR(1) coefficient for the complete signal of length 1500, then plot the power spectrum of the signal.

Note: The function `[h,w] = freqz(1,a,N)` returns the N-point frequency response vector, `h`, and the corresponding angular frequency vector, `w`, for the digital filter with AR coefficients in the vector `a`. Does this method capture the changes in frequency given by Equation (39)?

- b) Implement the CLMS algorithm to estimate the AR coefficient of the signal  $y(n)$ , see (??). At each time instant, compute the frequency spectrum of the signal using the `freqz` function with the coefficient estimates from the CLMS. Plot the time-frequency spectrum. (Hint: Use the code below.) Comment on the CLMS based spectrum estimate implemented in this part, compared to the stationary AR spectrum in Part a). [10]

```
for n = 1:N
    % Run complex-valued LMS algorithm to estimate AR coefficient â1(n)
    [h,w] = freqz(1, [1; -â1*(n)], 1024); % Compute power spectrum
    H(:, n) = abs(h).^2; % Store it in a matrix
end
% Remove outliers in the matrix H
medianH = 50*median(median(H));
H(H > medianH) = medianH;
% Plot time-frequency diagram
(Hint: To plot a time-frequency diagram, first plot a 3D graph with the entries of H in the z-axis, time instant in the x-axis, frequency w in the y-axis. Then, use the view(2) command to obtain a 2D color-heat map.)
```

### 3.3 A Real Time Spectrum Analyser Using Least Mean Square

Consider estimating a signal  $y(n)$  using a linear combination of  $N$  harmonically related sinusoids<sup>8</sup> as

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

where  $\hat{y}(n)$  is the estimate of the signal  $y(n)$  and  $w(k)$  are the unknown weights (in this case, the Fourier coefficients) that need to be estimated. Collecting all  $N$  estimates of the signal  $y(n)$  into a vector allows us to express this problem as

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

Within the least squares framework, the vector  $\mathbf{w}$  is found by minimising the sum of squared errors between the estimated signal  $\hat{y}(n)$  and true signal  $y(n)$ . This least squares problem is solved by minimising the cost function

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{\hat{y}}\|^2 = \min_{\mathbf{w}} \sum_{n=0}^{N-1} |y(n) - \hat{y}(n)|^2 \quad (40)$$

where  $\mathbf{y}$  is the vector containing the actual data  $\mathbf{y} = [y(0), y(1), \dots, y(N-1)]^T$ .

<sup>7</sup>First generate the frequency in (39), then integrate it using the MATLAB function `cumsum` to obtain the phase  $\phi(n)$ .

<sup>8</sup>Each sinusoid is a multiple of the frequency  $\frac{f_s}{N}$ .

- a) Show that the least squares (LS) solution for the problem in (40) is given by  $\mathbf{w} = (\mathbf{F}^H \mathbf{F})^{-1} \mathbf{F}^H \mathbf{y}$  and comment on its relationship to the discrete Fourier transform (DFT) formula. [5]
- b) Given the least squares interpretation of the DFT, in your own words, explain the Fourier transform in terms of the change of basis and projections. [5]

As seen in Part 3.2, a block-based solution can be implemented in an online fashion using adaptive filters. Similarly, treating the DFT as a least squares solution allows us to implement an adaptive version using the CLMS where the desired signal to the CLMS can be treated as the time-domain signal to be Fourier transformed and the input to the filter at time  $n$  is the complex phasor

$$\mathbf{x}(n) = \frac{1}{N} \left[ 1, e^{j\frac{2n\pi}{N}}, e^{j\frac{4n\pi}{N}}, \dots, e^{j\frac{2n(N-1)\pi}{N}} \right]^T \quad (41)$$

The filter weight vector  $\mathbf{w}(n) = [w_0(n), w_1(n), \dots, w_{N-1}(n)]^T$  is adapted using the CLMS algorithm (see Figure 8)

$$\begin{aligned} \hat{y}(n) &= \mathbf{w}^H(n) \mathbf{x}(n) \\ e(n) &= y(n) - \hat{y}(n) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu e^*(n) \mathbf{x}(n) \end{aligned} \quad (42)$$

Widrow et al. [8] showed that by setting the learning rate  $\mu = 1$ , and iterating over  $n$  from the initial conditions  $\mathbf{w}(0) = \mathbf{0}$ , the weight vectors  $\mathbf{w}(n)$  converge to the DFT solution.

- c) Implement the DFT-CLMS algorithm given in (42) for the frequency modulated signal from Part 3.2 a). Plot the magnitude of the weight vector  $\mathbf{w}(n)$  at every time instant to create a time-frequency diagram, see Part 3.2 b). Compare the DFT-CLMS to the adaptive AR-spectrum analyser in Part 3.2. Explain why the spectrum you obtained from the weights of the DFT-CLMS does not resemble to the true power spectrum? [15]
- d) Implement the DFT-CLMS for the EEG signal POz used in Part 1.2. To reduce computational burden, choose any segment POz of length 1200, e.g. POz(a:a+1200-1). Explain your observation about the time-frequency spectrum of the EEG signal. [5]

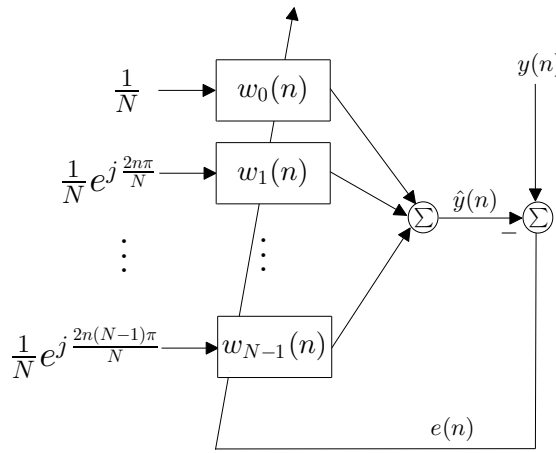


Figure 8: Block diagram of the DFT-CLMS.

## 4 From LMS to Deep Learning

**Aims:** Students will learn to:

- Understand the need to deal with bias, trend, and nonstationary and nonlinear behaviour in real-world data.
- Use their existing knowledge on stochastic gradient adaptation to design nonlinear learning structures.
- Learn how to develop Machine Intelligence architectures from a simple dynamical perceptron through to a deep multilayer neural network.
- Understand the importance of neural models for temporal data and learn how to implement, train, and assess the quality of neural networks based machine intelligence solutions.
- Understand the performance of temporal backpropagation networks and demonstrate the concepts of underfitting, overfitting, trade-off between complexity vs. accuracy, and optimal performance.
- Familiarise themselves with the concepts of block-based learning, training and test data, and learn about the importance of initial conditions in large and deep learning architectures.
- Understand the pivotal role of universal function approximation in Machine Intelligence, and learn through examples the foundations of expressive power, generalisation capability and the trade-off between the depth of a network and the speed of its response to outliers and anomalies.
- Understand the operation of NNs and CNNs as matched filters which aim to find patterns in data in a position invariant way.
- Gain insight into optimal (and more energy sustainable) ways to initialise CNNs, and to design fully-interpretable neural networks.

The general trend in machine intelligence is moving towards deep learning as a means of solving complicated classification and regression problems. This is conventionally introduced using simple problems which are time-independent and stationary. In Assignment 2, we saw that the LMS algorithm can be used for predicting non-stationary time-series in an online fashion. This section sheds further light onto the behaviour of the LMS algorithm for nonstationary streaming data, and demonstrates how it can be extended to quite general nonlinear regression problems [9]. We start from the prediction paradigm based on the time-series in Figure 9.

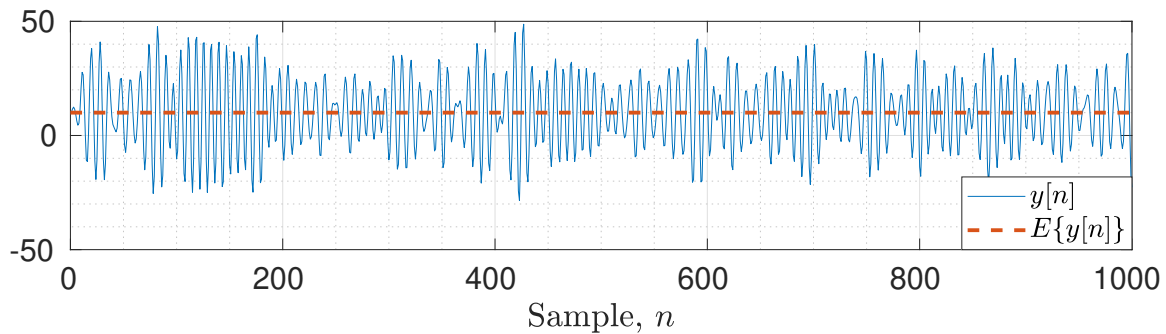


Figure 9: A non-stationary time-series (blue line) with non-zero mean (broken red line)

### 4.1 Neural Networks for Prediction

1. Load the time-series in Figure 9 from the ‘time-series.mat’ file found on Blackboard. Remove the mean from the time-series and use the LMS algorithm (with a learning rate of  $\mu = 1 \times 10^{-5}$ ) from Section 3 to predict  $y[n]$  from  $y[n-1], y[n-2], y[n-3]$  and  $y[n-4]$  (i.e. assume the data is generated using an AR(4) process). Plot the zero-mean version of the signal,  $y$ , against its one-step ahead prediction. Describe the behaviour of the output signal, and calculate the mean-square error (MSE) and prediction gain,  $R_p = 10 \log_{10} \left( \frac{\hat{\sigma}_y^2}{\sigma_e^2} \right)$ , where  $\hat{\sigma}_y^2$  is the variance of the output of LMS, and  $\sigma_e^2$  the variance of the prediction error. [10]
2. Typically, the generating process for the data is unknown and non-linear. Therefore, we can add a non-linearity (commonly referred to as an *activation function*) to the output of the LMS in order for the model to become more expressive, as in Figure 10 which shows a *dynamical perceptron*. Perform prediction of the zero-mean signal,  $y[n] - \mathbb{E}\{y[n]\}$ , using the dynamical perceptron and the  $\tanh$  function as a non-linearity. Plot the zero-mean signal against the output of the dynamical perceptron. Elaborate on whether this activation function is appropriate? [10]
3. The activation can be generalised by scaling the activation function, that is, changing its amplitude (i.e.  $a \cdot \tanh$ ). [10] What range of values for  $a$  would be appropriate for the data in Figure 9? Pick a value of  $a$  in your suggested range and repeat the one-step ahead prediction with your new proposed activation function. Plot the zero-mean signal and the new prediction. Comment on the prediction and MSE &  $R_p$  in comparison with the standard LMS.

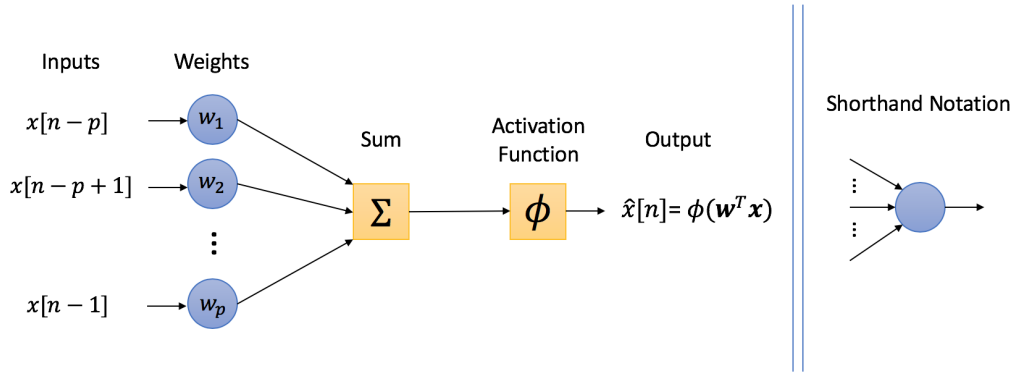


Figure 10: The dynamical perceptron model for predicting non-stationary time-series

4. Now, consider the original time-series in Figure 9 which exhibits a non-zero mean. In order to account for the mean automatically, we can add a bias input to our model. In other words, the output of the model can be mathematically described by:  $\phi(\mathbf{w}^T \mathbf{x} + b)$  where  $\phi(\cdot)$  is the activation function,  $\mathbf{w}$  are the weights of the model,  $\mathbf{x}$  is the input data and  $b$  is the bias. Hint: the bias can be implemented by considering the 'augmented' input to the algorithm as  $[1, \mathbf{x}]^T$ . Plot the original signal (non-zero mean) and the non-linear one-step ahead prediction with bias. Comment on the convergence of this algorithm. [10]
5. Since we are only performing a single weight update per time-step, it may take a lot of samples to converge to a reasonable prediction. One solution to this issue is to pre-train the weights by over-fitting to a small number of samples. Start with  $\mathbf{w}(0) = \mathbf{0}$  and use 100 iterations (also called *epochs*) to fit the model to the first 20 samples to yield  $\mathbf{w}_{\text{init}}$ . Then use  $\mathbf{w}_{\text{init}}$  as an initialisation to predict the entire time-series. Plot the original signal and the non-linear one-step ahead prediction, produced with bias and pre-trained weights. [10]
6. A simple activation function, like *tanh*, is not usually expressive enough to model real-world problems because of the highly nonlinear mapping from input to output. In order to tackle this issue, many simple "dynamical perceptron" models can be arranged together in order to create a 'deep network', as shown in Figure 11. Explain how the *backpropagation* algorithm can be used to train a deep network. [10]

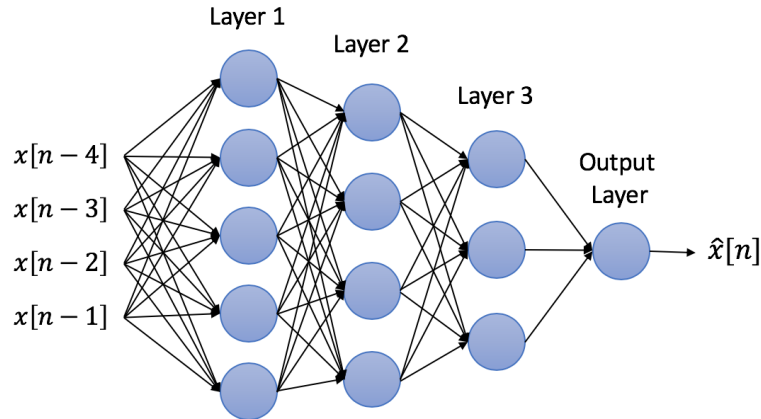



Figure 11: Example of a deep network with 3 hidden layers for predicting non-stationary time-series

There are many frameworks that allow for easy implementation of deep networks, however, the most popular frameworks (such as TensorFlow and PyTorch) have been developed in the Python programming language. An example of a highly nonlinear prediction problem is given at [https://github.com/am5113/ASPMI\\_DeepLearning](https://github.com/am5113/ASPMI_DeepLearning) (simply click the 'launch binder' button  to be able to run the code. Note: it may take a couple of minutes to load).

7. Train the deep network with default parameters (i.e. 4 hidden layers, 20,000 epochs, learning rate of 0.01 and noise power equal to 0.05). Comment on the test performance against epoch number for the deep network compared with a simple dynamical perceptron. [10]
8. Repeat the above for different values of noise power and comment on the drawbacks of using deep learning for prediction. [10]

## 4.2 Interpretable NNS: Convolutional Neural Network for Frequency Decomposition

**Background.** Convolutional neural networks (CNNs) aim to find patterns in signals by performing a cross-correlation between the learned weights (kernels) and the input signal. In this Assignment, you will illuminate the operation of CNNs in the context of spectral estimation, and gain insight into their full interpretability as matched filters [10].

### 4.2.1 Sinusoidal Signal Generation

Generate three sinusoidal signals of different frequencies in MATLAB or Python, using the following code:

```
t = ((2*pi)/100) : ((2*pi)/100) : 10*pi;
y1 = sin(t);
y2 = sin(0.5*t);
y3 = sin(4*t);
```

- Plot on the same graph the auto-convolution of signal  $y_1$ , and the convolution of signals  $y_1$  with  $y_2$ , and of  $y_1$  with  $y_3$ , and comment on the results. [5]
- Using only signals  $y_1$ ,  $y_2$  and  $y_3$ , and by using the convolution (conv) function (and optionally absolute value operator, mean operator, and max operator) as many times as you like, write a simple function to distinguish between  $y_1$ ,  $y_2$ ,  $y_3$ . The input to the function should be signal  $y$ , and the output should be the class 1, 2 or 3. Comment on any similarity with the operation of general CNNs. [5]

### 4.2.2 Sinewave Frequency Classification Analysis

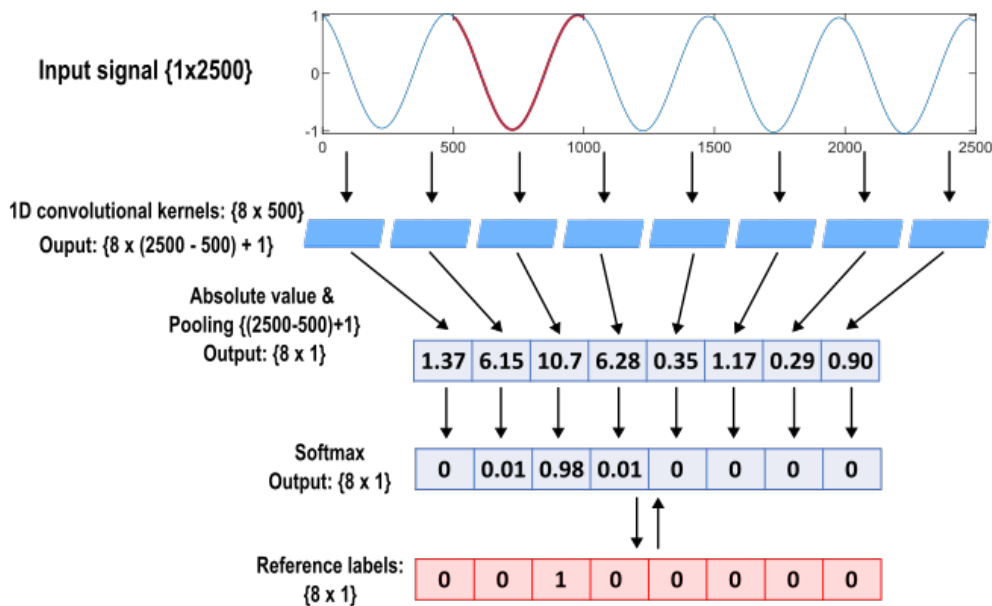


Figure 12: Network Architecture in Question 1.2

Implement the provided PyTorch code with the provided MATLAB generation code "training\_data\_generation.m" ([https://www.commsp.ee.ic.ac.uk/mandic/training\\_data\\_generation\\_1.zip](https://www.commsp.ee.ic.ac.uk/mandic/training_data_generation_1.zip)). The data consists of 2500 samples long sinewaves, at 8 different frequencies. For each sinusoid there is an 8 samples long binary label vector, with a 1 corresponding to the frequency of that specific signal and a 0 otherwise. The 8 different frequencies are:  $0, \cos(0.5t), \cos(t), \dots, \cos(3.5t)$ , where  $t$  has a period of  $2\pi$  every 500 samples. The block diagram of the architecture is provided in Figure 12, and the model architecture code is as follows:

```
import numpy as np
import torch
import torch.nn as nn
from sklearn.utils import shuffle
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
```

```

import scipy.io
import random

data_dct = scipy.io.loadmat('example_training_n2500_1freq_rescale_k8.mat')

#extract training data and labels
trainX = data_dct['trainX']
trainy = data_dct['trainy3']
trainX, trainy = shuffle(trainX, trainy)

#define training and model parameters
num_epochs = 200
batch_size = 30
n_in, n_out = 1, 8
kernel_size = 500
padding = 0
dilation = 1
stride = 1
learning_rate = 0.000001

L_in = trainX.shape[-1]
trainX = trainX.reshape((trainX.shape[0], 1, L_in))

# calculate the output length after convolution
L_out = int((L_in+2*padding-dilation*(kernel_size-1)-1)/stride+1)
# define the model using pytorch
class ConvNet1D(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Conv1d(n_in, n_out, kernel_size=kernel_size,padding=padding)
        self.layer2 = nn.MaxPool1d(L_out)
        self.act = nn.Softmax()

    def forward(self, x):
        out = self.layer1(x)
        out = torch.abs(out)
        out = self.layer2(out)
        out = self.act(out)
        return out

model = ConvNet1D()
# Loss and optimizer
criterion = nn.L1Loss() #nn.BCEWithLogitsLoss() #nn.L1Loss() #nn.MSELoss() #.
BCEWithLogitsLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = trainX.shape[0]

# transformation of data into torch tensors
trainXT = torch.from_numpy(trainX.astype('float32'))
trainyT = torch.from_numpy(trainy.astype('float32'))

##### create initialised weights for each of the 8 kernels
#Nt = kernel_size
#layer1_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
#layer2_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
#layer3_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
#layer4_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
#layer5_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))

```

```

#layer6_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
#layer7_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
#layer8_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))

#weights_init = np.concatenate((layer1_init,layer2_init,layer3_init,layer4_init,
    layer5_init,layer6_init,layer7_init,layer8_init),axis=0)
#weights_initT = torch.from_numpy(weights_init.astype('float32'))
#model.layer1.weight.data = weights_initT

loss_list = []
for epoch in range(num_epochs):
    correct_sum = 0
    correct_sum_test = 0
    for i in range(total_step//batch_size): # split data into batches
        trainXT_seg = trainXT[i*batch_size:(i+1)*batch_size, :, :]
        trainyT_seg = trainyT[i*batch_size:(i+1)*batch_size, :]

        # Run the forward pass
        outputs = model(trainXT_seg)

        #ensure outputs are the correct shape
        outputs = outputs.reshape((outputs.shape[0], 8))

        #calculate loss for the batch
        loss = criterion(trainyT_seg,outputs)

        # Backprop and perform Adam optimisation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    output_max = torch.argmax(outputs, 1, keepdim=False)
    reference_max = torch.argmax(trainyT_seg, 1, keepdim=False)
    count = torch.count_nonzero((output_max-reference_max)).cpu().detach()
    accuracy = 100*((batch_size-count)/batch_size)

    print("Epoch")
    print(epoch)
    print("Train_loss")
    print(loss)
    print("Accuracy_ (%) ")
    print(accuracy)
    loss_list.append(loss.item())

#plot the training accuracy per epoch
plt.title('Training_loss')
plt.ylabel('L1_Loss')
plt.xlabel('Epoch')
plt.grid(True)
plt.autoscale(axis='x', tight=True)
plt.plot(np.log(loss_list), label = "Training_loss")
plt.legend()
plt.show()

#### plot the trained kernel weights
kernels = model.layer1.weight.T.cpu().detach().numpy()[:,0,:]
fig, axs = plt.subplots(kernels.shape[1],1, figsize=(5,8), squeeze=True)
for i, ax in enumerate(axs):
    ax.plot(kernels[:,i])
plt.show()

```

- a) Plot the converged kernel weights after 200 epochs, based on the code snippet provided. Comment the interpretability of the so produced kernel weights. [5]
- b) Run the model first with average pooling (AvgPool1d), and then with max pooling (MaxPool1d). Comment on the differences in L1loss over the first 50 epochs (if any exist), and explain why such differences may arise based on the results of Section 1(a). [5]

### 4.2.3 Kernel Initialization Impact on Convergence

Using the code snippet provided, initialise each of the 8 kernels with a function of your choice. Note that the only code to be changed should be : [10]

```
|| np.zeros((1, kernel_size))
```

Comment on the reasoning behind your initialisation and on how this induces any changes in the convergence of loss function.

```
##### Replace this comment in the above code to your answer
##### create initialised weights for each of the 8 kernels
Nt = kernel_size
layer1_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer2_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer3_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer4_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer5_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer6_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer7_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))
layer8_init = np.reshape(np.zeros((1, kernel_size)), (1,1,kernel_size))

weights_init = np.concatenate((layer1_init,layer2_init,layer3_init,layer4_init,
                                layer5_init,layer6_init,layer7_init,layer8_init),axis=0)
weights_initT = torch.from_numpy(weights_init.astype('float32'))
model.layer1.weight.data = weights_initT
```



## 5 Tensor Decompositions for Big Data Applications

**Aims:** Students will learn to:

- Understand the consequences of the *Curse of Dimensionality* associated with Big Data applications.
- Familiarise themselves with the basics of multi-linear algebra and multi-way processing of high-dimensional data.
- Learn about the value of data re-shaping, tensorisation, and the basic tensor-matrix operations, such as contraction products and factor matrices.
- Extend the concept of matrix rank to the concept of higher-order tensor rank, and demonstrate the benefits of the flexibility of tensor algebra over the “flat-view” two-dimensional matrix algebra.
- Verify the super-compression ability of tensors, and gain hands-on experience of tensor decompositions.
- Became familiar with the Tucker Decomposition, an example of Higher-Order Singular Valued Decomposition (HO-SVD), and demonstrate the “*Blessing of Dimensionality*” associated with tensor decompositions.
- Treat a colour image as a third-order tensors and demonstrate how the varying tensor rank can be used to extract information from images at different scales.
- Understand the physically meaningful way, in a hands-on fashion, in which tensor decompositions perform dimensionality reduction and Big Data analytics.
- Gain intuition behind the connection between tensor networks and deep learning architectures.

**Background.** In this part of the coursework, you will familiarise yourselves with multi-dimensional arrays, their efficient representations (decompositions) and their super-compression properties. You will learn the basics of tensor multilinear algebra and will gain hands-on experience with commonly used computational methods for performing tensor decompositions. You will be able to demonstrate how by exploring the structure in multi-way data, it is possible to represent even Terabyte or larger volumes of data in a computationally affordable way. You will also be able to compute efficiently various data association and dimensionality reduction strategies on such huge data volumes [11].

Throughout this assignment you will be using popular Python libraries and our own tensor decomposition and manipulation software, called the Higher Order Tensor Train Toolbox (HOTTBOX).

All questions in this Assignment are presented in the form of Jupyter notebooks, a very convenient web-based interactive computation environment.

1. In order to get started, please copy-paste the following URL [30]  
<https://github.com/IlyaKisil/dpm-coursework>  
to your web browser, and follow instructions therein. The whole Assignment will need to be solved using this web-link, where the illustrative practical examples will be provided, too.

Welcome to the wonderful world of Big Data!

## References

- [1] M. Hayes, *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, 1996.
- [2] A. Benveniste, M. Métivier, and P. Priouret, *Adaptive algorithms and stochastic approximations*. Springer Publishing Company, Incorporated, 2012.
- [3] W.-P. Ang and B. Farhang-Boroujeny, “A new class of gradient adaptive step-size LMS algorithms,” *IEEE Transactions on Signal Processing*, vol. 49, no. 4, pp. 805–810, 2001.
- [4] V. Mathews and Z. Xie, “A stochastic gradient adaptive filter with gradient adaptive step size,” *IEEE Transactions on Signal Processing*, vol. 41, no. 6, pp. 2075–2087, 1993.
- [5] D. Mandic, “A generalized normalized gradient descent algorithm,” *IEEE Signal Processing Letters*, vol. 11, no. 2, pp. 115–118, 2004.
- [6] D. Mandic, S. Javidi, S. Goh, A. Kuh, and K. Aihara, “Complex-valued prediction of wind profile using augmented complex statistics,” *Renewable Energy*, vol. 34, no. 1, pp. 196 – 201, 2009.
- [7] D. P. Mandic and V. S. L. Goh, *Complex Valued Nonlinear Adaptive Filters: Noncircularity, Widely Linear and Neural Models*. John Wiley & Sons, 2009.
- [8] B. Widrow, P. Baudrenghien, M. Vetterli, and P. Titchener, “Fundamental relations between the LMS algorithm and the DFT,” *IEEE Transactions on Circuits and Systems*, vol. 34, no. 7, pp. 814–820, 1987.
- [9] D. P. Mandic and J. A. Chambers, *Recurrent Neural Networks for Prediction: Algorithms, Architectures, and Stability*. John Wiley & Sons, 2001.
- [10] L. Stankovic and D. P. Mandic, “Convolutional neural networks demystified: A matched filtering based approach,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 53, no. 6, pp. 3614–3628, 2023.
- [11] A. Cichocki, D. P. Mandic, C. Caiafa, A.-H. Phan, G. Zhou, Q. Zhao, and L. D. Lathauwer, “Tensor decompositions for signal processing applications: From two-way to multiway component analysis,” *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.