# 1 | HW01 - Theoretical tools, methods adopted and code kernel

## 1.1. Noise generation

For the first point we need to estimate the covariance from a set of L samples of a generated noise. I generated the correlated noise starting from an uncorrelated Gaussian noise $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the given covariance matrix $[\mathbf{C_{ww}}]_{i,j} = \sigma_w^2 \rho^{|i-j|}$, using the Cholesky decomposition to obtain a lower triangular matrix $Q$ from $\mathbf{C_{ww}} = \mathbf{QQ^T}$. The generated noise is hence $\mathbf{w} = \mathbf{Qg}$.

I used the formula $\hat{\mathbf{C}} = \frac{1}{L-1} \sum_{k=1}^{L} (\mathbf{w_k} - \overline{\mathbf{w}}_\mathbf{k})(\mathbf{w_k} - \overline{\mathbf{w}}_\mathbf{k})^T$ to estimate the covariance matrix over L realizations of the generated noise.

Then, as requested, I computed the MSE (Mean Squared Error) for different sample sizes N and for different values of correlation coefficients $\rho$:

$MSE = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\hat{\mathbf{C}}_{i,j} - \mathbf{C_{ww}}_{i,j})^2$.

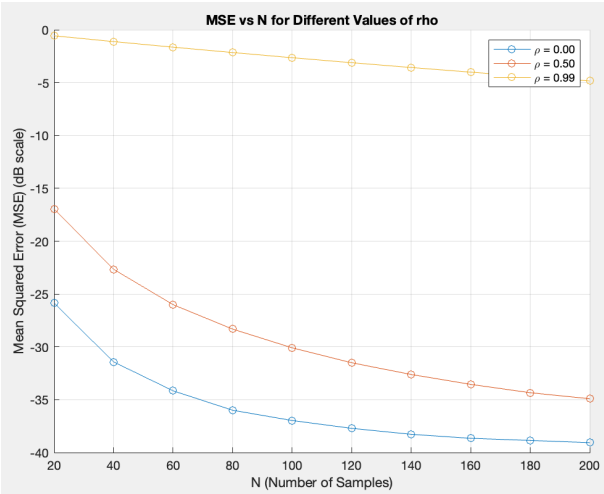



Figure 1.1: MSE vs. N for different values of $\rho$.

As expected (see the plot in figure 1.1) the MSE reduces when we increase the number of samples N due to the fact that I have more samples to estimate $\hat{\mathbf{C}}$ and so more accuracy in the estimation (more similar to $\mathbf{C_{ww}}$). Instead, the MSE increases as we increase the value of the correlation coefficient $\rho$ because the samples are more correlated between them (less independent) and that makes it harder to estimate the covariance matrix.

## 1.2. Frequency estimation

The goal of this section was to evaluate the MSE on frequency estimation vs SNR ($SNR = \frac{a_0^2/2}{\sigma_w^2}$) with CRB.

### 1.2.1. Filter $h[n] = \delta[n]$

For this section we assume that the filter is removed ($h[n] = \delta[n]$) and the noise is uncorrelated $\mathbf{C_{ww}} = \sigma_w^2 \mathbf{I}$ ($\rho = 0$). We need to estimate the frequency of a sinusoid with a uniformly distributed phase $\phi_0 \sim \mathcal{U}(0, 2\pi)$ at different N, $\omega$, SNR.

The amplitude $a_0$ of the sinusoid $x[n] = a_0 cos(\omega n + \phi_0)$ can be obtained from the SNR as $a_0 = \sqrt{2SNR\sigma_w^2}$, where $\sigma_w$ is set to 1.

Then, I added the uncorrelated noise $w \sim \mathcal{N}(0, \sigma_w^2 \mathbf{I})$ (Gaussian with zero mean) to the sinusoid $x[n]$ to obtain the signal $y[n] = x[n] + w[n]$

To estimate the frequency I used FFT (Fast Fourier Transform) with zero padding and parabolic estimation. I estimated the power spectrum $S[k]$ of the noisy signal through FFT $Y[k] = \sum_{n=0}^{N-1} y[n] e^{-j\frac{2\pi kn}{N}}$ as $S[k] = \frac{|Y[k]|^2}{N}$. The peak of the power spectrum corresponds

to the dominant frequency of the signal.

To achieve a better FFT resolution I extended $y[n]$ with zeros to length M>N (zero padding) so that $\Delta f = \frac{2\pi}{M} < \frac{2\pi}{N}$.

Now, I need to find the value that maximizes the power spectrum $k_{max} = argmax_k S[k], k = 2, 3, ..., M/2$ (k=1 is removed to skip the DC part). To improve the precision I applied the parabolic interpolation around the max:

$\delta = \frac{1}{2}\frac{S[k_{max}+1]-S[k_{max}-1]}{S[k_{max}+1]+S[k_{max}-1]-2S[k_{max}]}$

$\hat{\omega}_k = \Delta f(k_{max} + \delta) = \frac{2\pi}{M}(k_{max} + \delta)$

The MSE across L realizations of the sinusoidal signal is $MSE = \frac{1}{L}\sum_{i=1}^{L}(\hat{\omega}_i - \omega)^2$.

The CRB for frequency estimation is given by the formula $CRB = \frac{12\sigma_w^2/a_0^2}{N(2N-1)(N-1)} = \frac{6}{N(2N-1)(N-1)}\frac{1}{SNR}$.

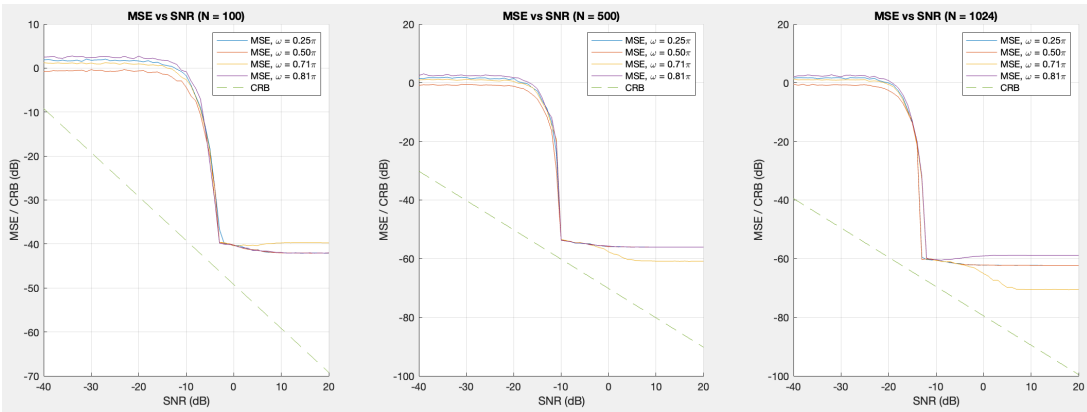



Figure 1.2: MSE vs. SNR for different $\omega$ and CRB.

For larger N, the MSE decreases because more samples improve estimation accuracy.

At low SNR the MSE deviates from the CRB due to the noise dominating the signal, at high SNR the MSE approaches the CRB because the estimation becomes more accurate with less noise.

The MSE trend is pretty much the same for every frequency $\omega$.

### 1.2.2. Filter $h[n] \neq \delta[n]$

This time we need to take into account a filter $h[n]$ ($H(z) = \frac{2}{1+0.9z^{-1}}$, AR(1) and min phase) so that $y[n] = x[n] * h[n] + w[n]$. We have:

$Y'(z) = H(z)X(z) = \frac{2}{1+0.9z^{-1}}X(z) \longrightarrow Y'(z) + 0.9z^{-1}Y'(z) = 2X(z)$

$y'[n] = 2x[n] - 0.9y[n-1]$ and $y[n] = y'[n] + w[n]$

Also, the CRB changes due to filtering: $CRB = \frac{12}{N(2N-1)(N-1)}\frac{\sigma_w^2}{b_0^2}$ where $b_0 = a_0|H|$ . Then, everything is the same as before.

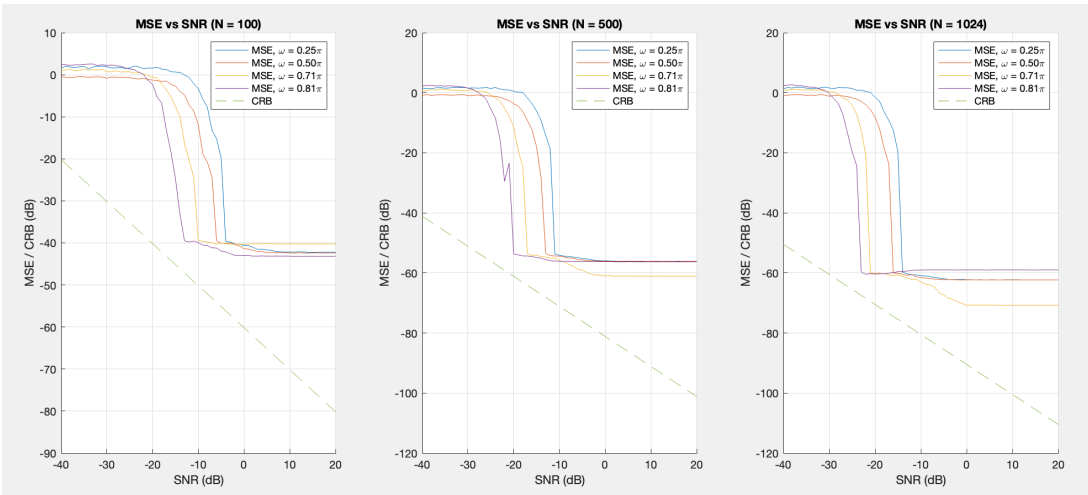



Figure 1.3: MSE vs. SNR for different $\omega$ and CRB (filtered signal).

We notice that the MSE is generally lower than before (without the filter) because the Auto Regressive AR(1) filter enables to clean the noise that comes from the signal generation and enhance the frequency of interest making it easier to estimate.

For higher frequencies the MSE is lower, that's because the parabolic interpolation become more reliable with sharper peaks in the frequency spectrum (higher $\omega \rightarrow$ sharper spectrum). This time the difference between $\omega$ is more evident because the filtered signal is less affected by noise thanks to the filtering.

### 1.2.3. Covariance is correlated

For this point I needed to repeat the exercises above but with correlated noise ($[\mathbf{Cww}]_{i,j} = \sigma_w^2\rho^{|i-j|}$ with $\rho = 9/10$).

I generated the noise using Cholesky decomposition of the covariance matrix

$\mathbf{C_{ww}} = \mathbf{QQ^T} \longrightarrow \mathbf{w} = \mathbf{Qg}$ with $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Lastly, I redefined the CRB for the correlated noise as (from the general Fisher-information formula for a Gaussian model with covariance $C_{ww}$): $CRB = \frac{2\sigma_w^2}{a_0^2}\frac{1}{\sum_{i,j}^{N-1} n_i[C_{ww}^{-1}]_{ij}n_j}$ .

And for the filtered signal with correlated noise: $CRB = \frac{2\sigma_w^2}{b_0^2} \frac{1}{\sum_{i,j}^{N-1} n_i [C_{ww}^{-1}]_{ij} n_j}$ where $b_0 = a_0|H|$ .

From the results we clearly see that the MSE is worsened both without (figure 1.4) and with (figure 1.5) filtering. This is due to the correlated noise that makes the estimation of the frequency a lot harder than before. Anyway, the filtering of the signal makes the estimation better (the MSE goes down a bit than without filtering).

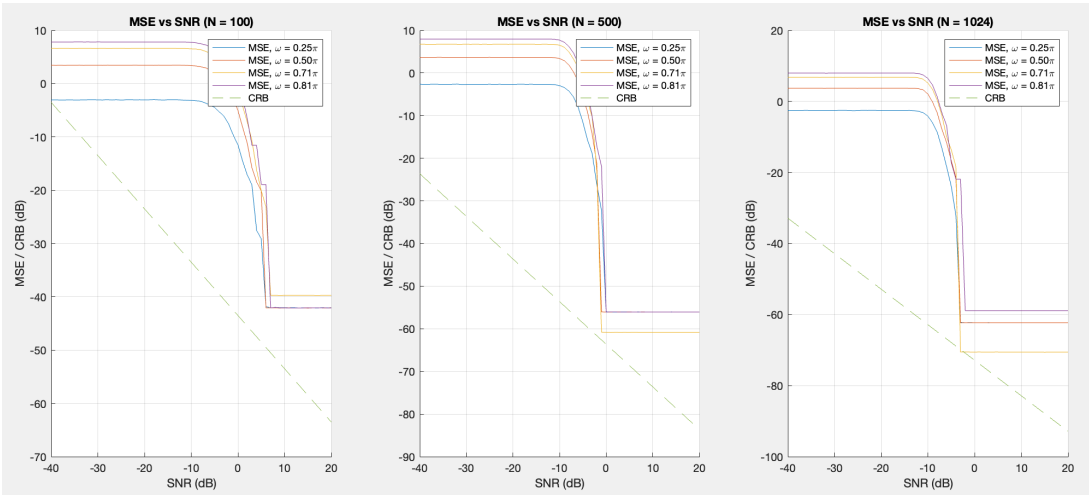



Figure 1.4: MSE vs. SNR for different $\omega$ and CRB (correlated noise).

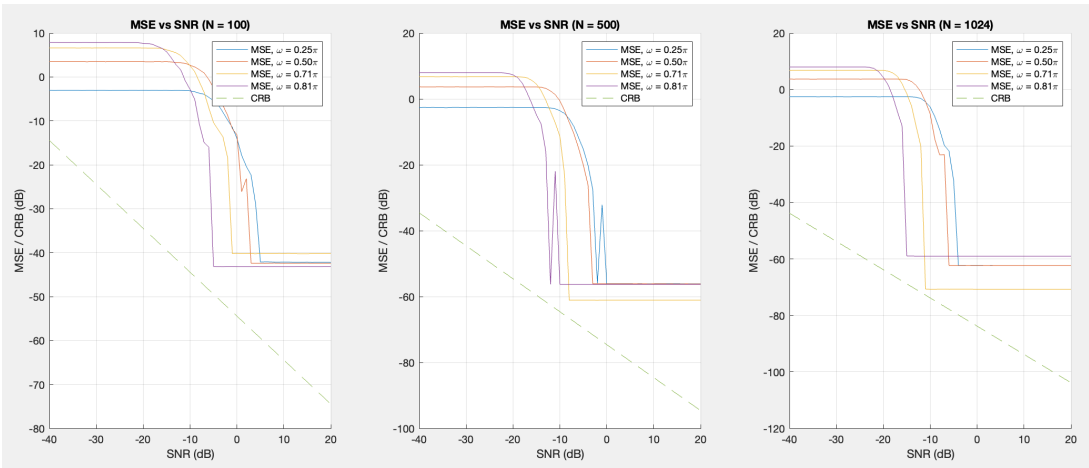



Figure 1.5: MSE vs. SNR for different $\omega$ and CRB (correlated noise and filtered signal).

## 1.3. Frequency modulation

Now the request was to generate a frequency modulated sinusoid $x[n] = a \cdot cos(\gamma n^2 + \phi) + w[n]$ and estimate the instantaneous frequency $\omega(n) = 2\gamma n$, knowing that $\omega(0) = \frac{\pi}{8}$ and $\omega(N) = \frac{3\pi}{4}$. So to compute $\gamma$: $\omega(N) - \omega(0) = 2\gamma(N - 0) = 2\gamma N = \frac{3\pi}{4} - \frac{\pi}{8}$ that leads to $\gamma = \frac{\frac{3\pi}{4} - \frac{\pi}{8}}{2N}$. Calling $\varphi = \gamma n^2 + \phi$, the instantaneous frequency for discrete signals is computed as $f_{inst}(n) = \frac{1}{2\pi}\Delta\varphi(n) \cdot f_s = \frac{1}{2\pi}(\varphi(n+1) - \varphi(n)) \cdot f_s$.

Then I had to estimate the signal through the MLE. To do so, I need to find the value of $\gamma$ that minimizes the cost function $J$(equivalent to maximize the likelihood function):

$\hat{\gamma} = argmin_\gamma J(\gamma) = argmin_\gamma(\sum_{n=0}^{N-1}(x - a \cdot cos(\gamma n^2 + \phi))^2)$. (see figure 1.6)

The reconstructed signal is given by $x_{estimated} = a \cdot cos(\hat{\gamma} n^2 + \phi)$ (comparison between estimated signal and noisy signal in figure 1.7).

The true gamma was 0.00004463 while the estimated gamma was 0.00004449 (very close, good estimate).

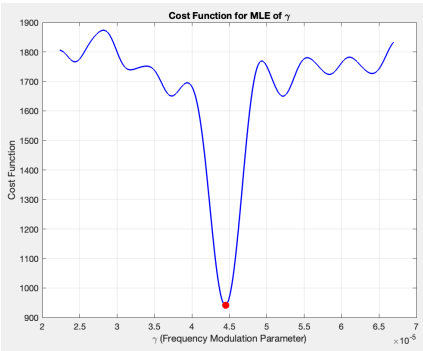



Figure 1.6: Cost function for different values of $\gamma$.

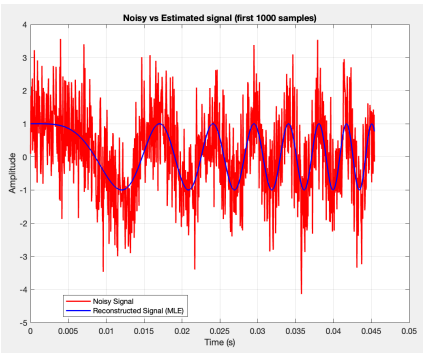



Figure 1.7: Estimated signal(blue) and noisy signal(red).

# 2 Conclusions

This project analyzed frequency estimation under different conditions, considering noise characteristics, filtering, and frequency modulation. The results confirmed that increasing the number of samples improves covariance estimation, while higher noise correlation negatively impacts accuracy. Applying an autoregressive filter helped reduce noise, leading to better frequency estimates. The FFT-based method with parabolic interpolation showed good accuracy, approaching the theoretical Cramer-Rao Bound at high SNR. For frequency modulation, the instantaneous frequency was estimated using phase differences, and the signal was reconstructed through maximum likelihood estimation, achieving reliable results despite noise.

As an improvement, implementing adaptive filters could allow dynamic noise reduction based on real-time estimation of noise characteristics.

The methods studied here are applicable in various engineering domains, including wireless communications (for example channel estimation), biomedical signal processing, and radar signal processing.

Further research could analyze the impact of different noise models, such as non-Gaussian or colored noise, and explore alternative modulation schemes beyond simple frequency modulation.

# 1 | HW02 - Theoretical tools, methods adopted and code kernel

## 1.1. Point a)

First of all, I modeled the system to better understand and visualize what is happening 1.1.

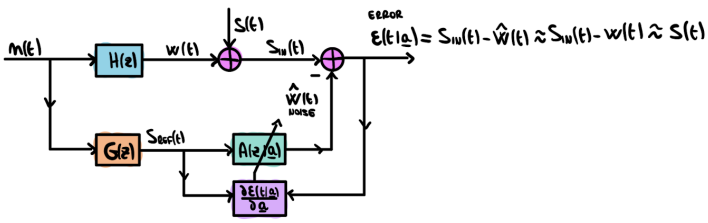



Figure 1.1: Block diagram of the system.

In order to obtain the signal, I need to evaluate the best possible estimation of the noise $\hat{w}$ because if this estimation would be equal to the noise itself, the error would correspond to the signal without the noise. The estimation of the noise is done combining the reference signal of the noise and the adaptive filter.

### 1.1.1. NLMS algorithm

The filter coefficients are investigated by minimizing the MSE:

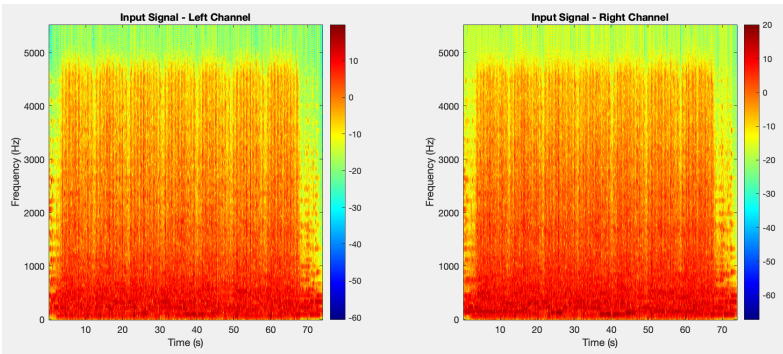
$J(\mathbf{a}) = \mathbb{E}[\varepsilon^2[k|\mathbf{a}]] = \mathbb{E}[(\mathbf{s}_{in}[k] - \mathbf{a}^T\mathbf{s}_{ref}[k])^2]$

Using the LMS (Least Mean Square), the gradient for the update of the filter coefficients $\mathbf{a}$ is based on the instantaneous error ( only 1 sample, not the mean):

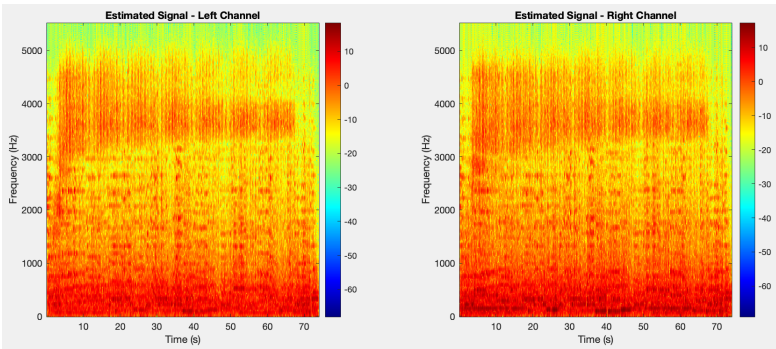
$\frac{\partial J(\mathbf{a}[k])}{\partial \mathbf{a}[k]} \simeq \frac{\partial}{\partial \mathbf{a}[k]}(\mathbf{s}_{in}[k] - \mathbf{a}^T\mathbf{s}_{ref}[k])^2 = 2(\mathbf{s}_{in}[k] - \mathbf{a}^T\mathbf{s}_{ref}[k])\mathbf{s}_{ref}[k] = 2\varepsilon[k]\mathbf{s}_{ref}[k]$

$\mathbf{a}[k+1] = \mathbf{a}[k] + \mu \frac{\varepsilon[k]\mathbf{s}_{ref}[k]}{||\mathbf{s}_{ref}[k]||^2}$ where $\mu$ is the step size.

I plotted a spectrogram of the result to understand if the LMS worked (figure 1.2).




(a)




(b)

Figure 1.2: Spectrograms of the Input signal Sin (a) and the estimated signal Sest (b), for the left and right channel.

As we can see, the spectra of the signal were better but still not very clean.

### 1.1.2. Cut-Off and $\mu$ adjusting

The best possible solution was now to cut off the high frequencies of the estimated signal where the most of the noise was concentrated. I identified which was the region of the spectra to cut-off(from 2500Hz to 4500Hz) and calculated the corresponding indices in the frequency domain (because it is a discrete sequence so I need to approximate to the closest values of $f_{high}$ and $f_{low}$). Then I computed the FFT (Fast Fourier Transform) to transpose the signal in the frequency domain, cut off the frequency window from both channels, and bring the signal back to the time domain using the IFFT(Inverse Fast Fourier Tranform) taking only the real part of the signal.

I also adjusted "manually" the value of the $\mu$ for both channels using different values

for the left and right channels and observing the plots of the MSE computed, trying to balance the value of the step. Indeed, if the step is too small the time of convergence would be too big, while if the step is too big the error between the final MSE found and the minimum MSE would be non-negligible.

Eventually, the spectrogram shows a satisfying result, as well as the sound played in Matlab.

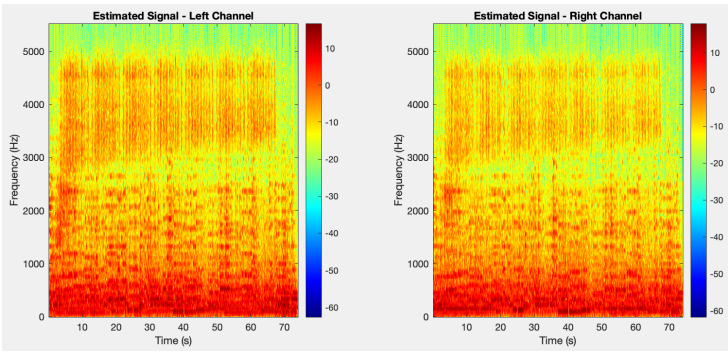
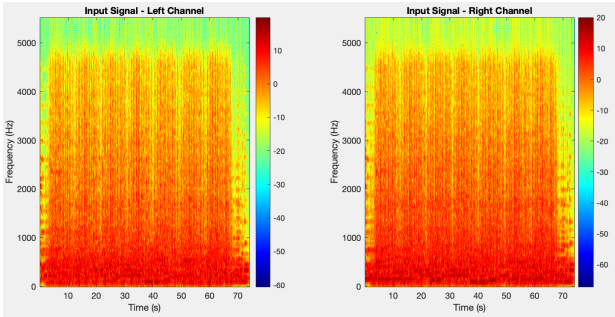


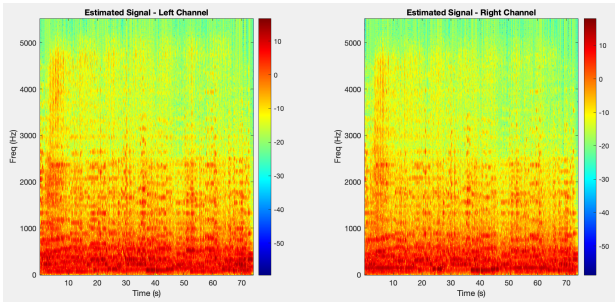

Figure 1.3: Spectrograms of the estimated signal with the cutoff of the high frequencies, for the left and right channel.

## 1.2. Point b)

For this point, I just repeated the procedure of point a), but this time I took both the channels as reference for the estimation of the left and the right channel. In figure 1.4, as we can see in the spectrogram, the result is even better than point a). I adjusted the values for the step sizes $\mu$ and window length, and chosen the optimal values resulting $\mu = 0.04$ for the left channel, $\mu = 0.05$ for the right channel and $WindowLength = 46$.




(a)




(b)

Figure 1.4: Spectrograms of the Input signal Sin (a) and the estimated signal Sest (b), for the left and right channel.

## 1.3. Point c)

Also for this point I used the same procedure as point b). In figure 1.5 we can see the result using $\mu = 0.015$ for the left channel and $\mu = 0.02$ for the right channel.

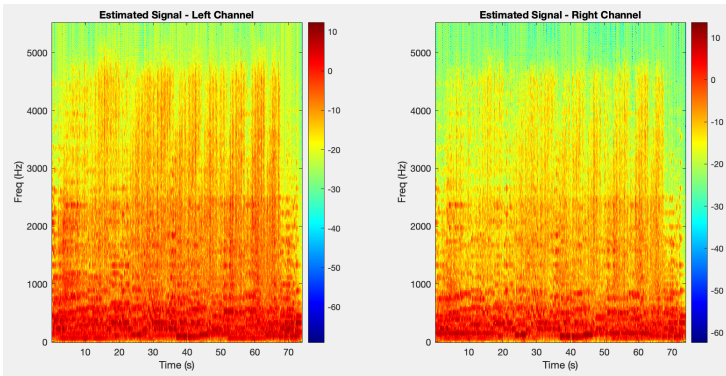



Figure 1.5: Spectrograms of the estimated signal for the left and right channel.

## 1.4. Point d)

From this point on, the problem changes and the LMS and RLS methods could no longer be applied. In the spectrum of the noisy signal (figure 1.6), we can clearly see sinusoidal noise that appears as prominent peaks in the frequency spectrum.

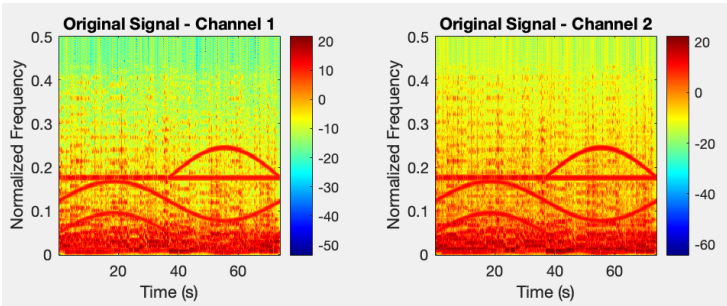



Figure 1.6: Spectrograms of the noisy signal for the left and right channel.

These peaks correspond to dominant frequencies in the signal. The method applied was to identify and remove the three most prominent peaks. To do so I applied a Notch filter, a band-stop filter that attenuates a narrow frequency range while allowing other frequencies to pass. The transfer function of the Notch filter is $H(z) = \frac{1-2cos(\omega_0)z^{-1}+z^{-2}}{1-2rcos(\omega_0)z^{-1}+r^2z^{-2}}$, where $\omega_0$ is the notch frequency (where the filter is centered) and $r$ is a parameter that controls the filter width. Then I applied a zero-phase filtering in order to have no phase distortion after the notch filter (filter forward, then reverse the result and filter again). The resulted signal in figure 1.7, shows that the noise (sinusoids) has been erased thanks to the method applied.

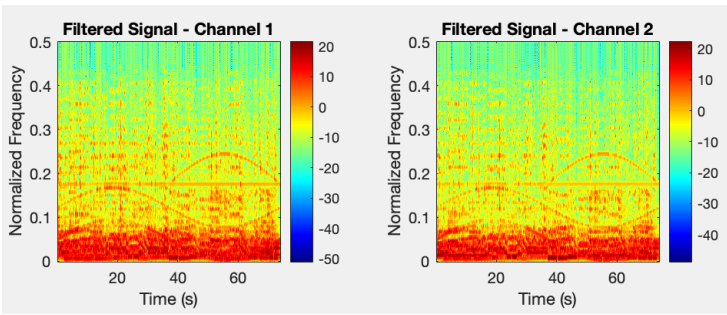



Figure 1.7: Spectrograms of the estimated signal for the left and right channel.

Since there was still some noise in the higher frequencies I tried to use the cut-off technique used before (figure 1.8).

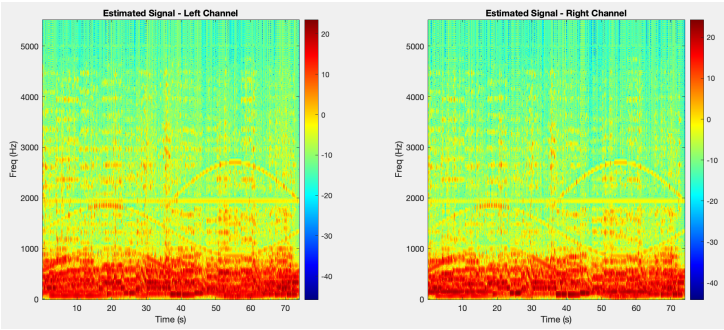
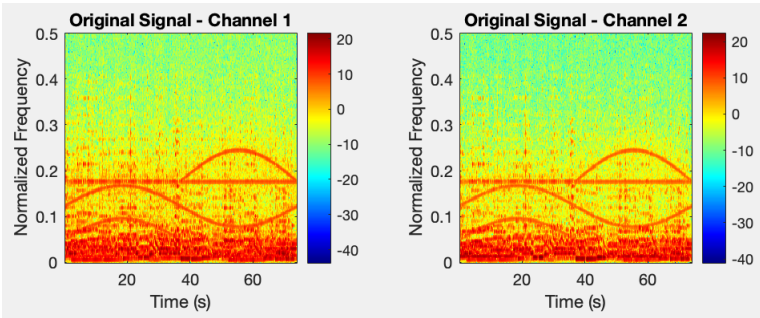


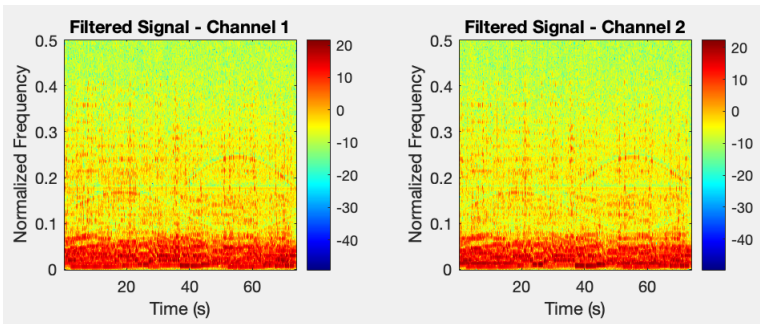

Figure 1.8: Spectrograms of the estimated signal for the left and right channel after high frequencies cut-off of the estimation.

## 1.5. Point e)

Since the problem was similar as before, I applied the same procedure as in point d). The results are pretty good and are shown in figure 1.9.




(a)




(b)

Figure 1.9: Spectrograms of the Input signal Sin (a) and the estimated signal Sest (b), for the left and right channel.

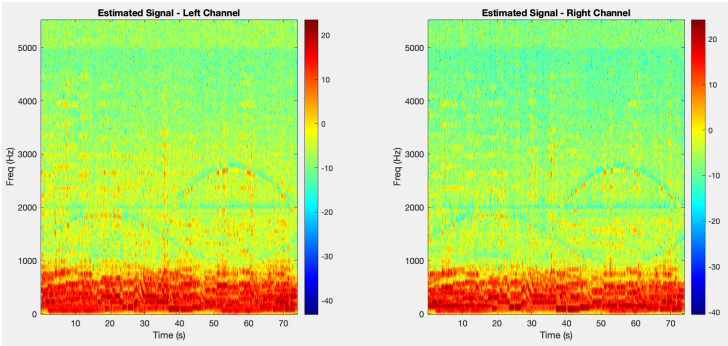



Figure 1.10: Spectrograms of the estimated signal for the left and right channel after high frequencies cut-off of the estimation.

# 2 Conclusions

In conclusion, we can say that the results obtained with LMS algorithm were good but could be improved (specifically for point c)), while the Notch filtering procedure for the last two points was very successful looking at the spectrograms.

Beyond audio, the same principles could be applied to financial predictions, biomedical signal processing and other fields.

# 1 | HW03 - Theoretical tools, methods adopted and code kernel

## 1.1. Part 1 - Point 1)

For this point, I need to derive an estimator of every value $\hat{x}_k(t)$ using linear prediction from $x_k(t)$ by increasing the prediction length and the prediction step. Firstly, I removed the mean ($\mu_k = \frac{1}{T}\sum_{t=1}^{T} x_k(t)$) from all the series in order to have zero-mean series. I computed the autocorrelation $r_{xx} = \frac{1}{T}\sum_{t=1}^{T} x[t]x[t+l]$ where $l = 0, 1, 2, ..., L = p + h$ (p: prediction length, h: prediction step). For the prediction estimate, I used the Yule-Walker equation $r_{xx}[l] + \sum_{i=1}^{p} a_i r_{xx}[n+1-i] = 0$ with $l = 1, 2, ..., p$. To compute the coefficients:

$$\mathbf{a} = -\mathbf{R}_p^{-1}\mathbf{r} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix} = -\begin{bmatrix} r_{xx}(0) & r_{xx}(1) & \cdots & r_{xx}(p-1) \\ r_{xx}(1) & r_{xx}(0) & \cdots & r_{xx}(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_{xx}(p-1) & r_{xx}(p-2) & \cdots & r_{xx}(0) \end{bmatrix}^{-1} \begin{bmatrix} r_{xx}(1) \\ r_{xx}(2) \\ \vdots \\ r_{xx}(p) \end{bmatrix}.$$

Given the state vector at time t before forecasting $\mathbf{y}^{(0)} = \begin{bmatrix} x[t-1] \\ x[t-2] \\ \vdots \\ x[t-p] \end{bmatrix}$, the prediction at step h is given by $\mathbf{y}^{(h)} = -\mathbf{a}^T\mathbf{y}^{(h-1)}$ (so that $\hat{x}[t+h] = y^{(h)}(1)$). To choose the best prediction length p and prediction step h, I plotted the MSE vs p vs h (figure 1.1) and observed which was the minimum of the 3D graph. The bigger the prediction length and the smaller the prediction step, the better is the prediction.

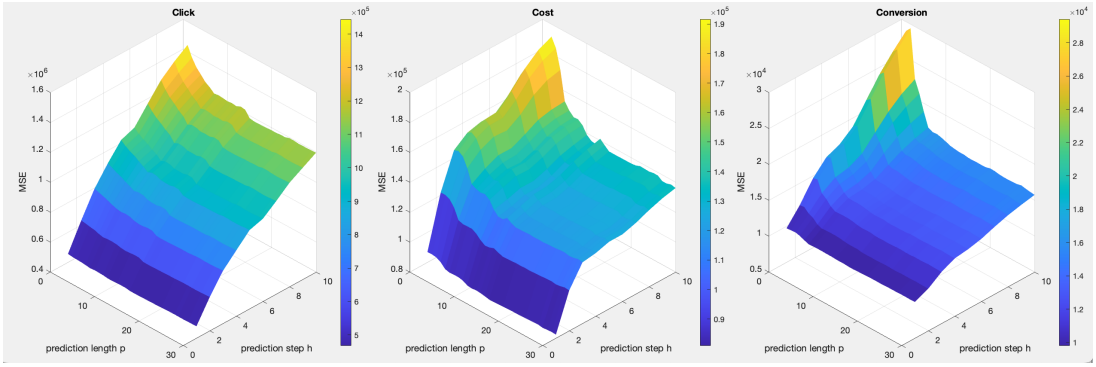



Figure 1.1: MSE vs. prediction length p vs. prediction step h.

Using the information of the MSE plot, I predicted the series using p=30 and h=1. As we can see in figure 1.2, for K=20, the prediction estimate is very close to the actual values of the trace.

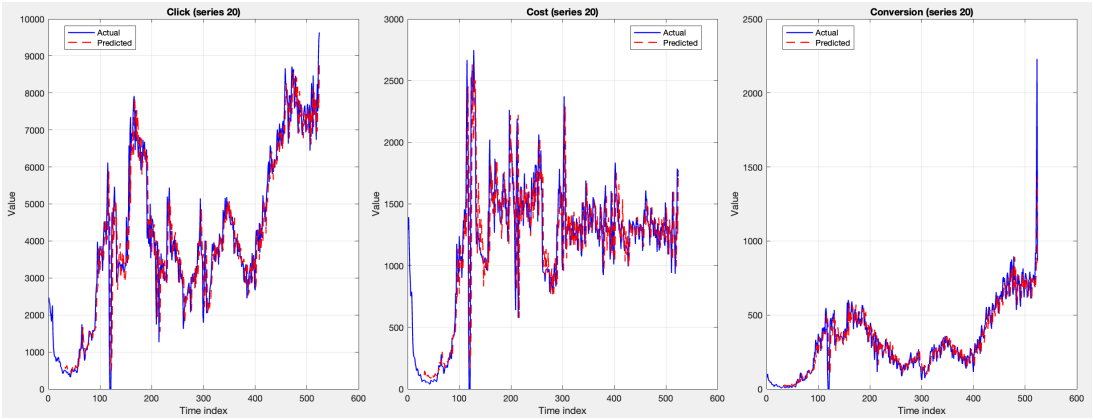



Figure 1.2: Actual and predicted values for the series K=20.

## 1.2. Part 1 - Point 2)

For this point, we needed to estimate the trace $x_i(t)$ using all the other traces: $\hat{x}_i(t) = f_i[x_1(t), ..., x_k(t), ..., x_K(t)|k \neq i]$

### 1.2.1. Point 2a)

I used the Least Squares method,
minimizing the sum of squared errors $argmin(\sum_{t=1}^{T}(x_i(t) - \mathbf{w}^T\mathbf{x}_{-i}(t))^2)$, that leads to

$$\mathbf{w}_i = (\mathbf{X}_{-i}^T \mathbf{X}_{-i})^{-1} \mathbf{X}_{-i}^T \mathbf{y}_i \text{ where } \mathbf{X}_{-i} = \begin{bmatrix} \mathbf{x}_{-i}(1)^T \\ \mathbf{x}_{-i}(2)^T \\ \vdots \\ \mathbf{x}_{-i}(T)^T \end{bmatrix},$$

$$\mathbf{x}_{-i}(t) = \begin{bmatrix} x_1(t) & \cdots & x_{i-1}(t) & x_{i+1}(t) & \cdots & x_K(t) \end{bmatrix}^T,$$

$$\mathbf{y}_i = \begin{bmatrix} x_i(1) & \cdots & x_i(T) \end{bmatrix}^T.$$

The prediction is computed as $\hat{x}_i(t) = \mathbf{w}_i^T \mathbf{x}_{-i}$ .

As we can see in figure 1.3, the result is worse than the method in point 1.

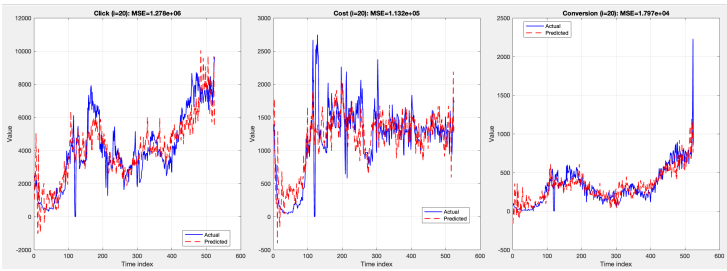



Figure 1.3: Actual and predicted values for the series K=20.

Calculating the MSE for each trace (for every KPI), it is possible to see in figure 1.4 that some of the values are high (prediction not so correct). The reason could be that some of the traces that I used for the prediction was not actually useful, leading to a worse estimate.

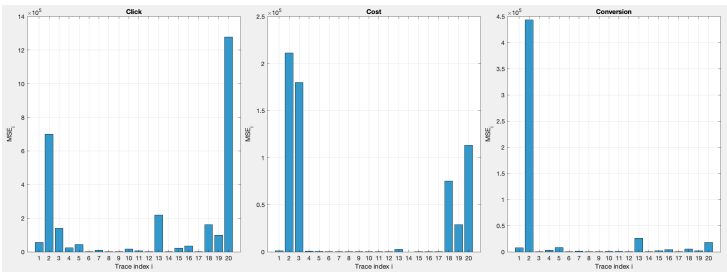



Figure 1.4: MSE computed for different traces and KPIs.

### 1.2.2. Point 2b)

Since maybe not all traces are useful for the estimate of the trace of interest (here we focus on the trace K=20), I computed the MSE every time I dropped one of the traces (for each KPI) in figure 1.5. Each of the trace is useful for a good estimate of the trace K=20. Ones of the most important traces for the best estimate of the trace (the ones

that increase the most the MSE if dropped) are: trace 4 for Click, trace 19 for Cost and trace 5 for Conversion.

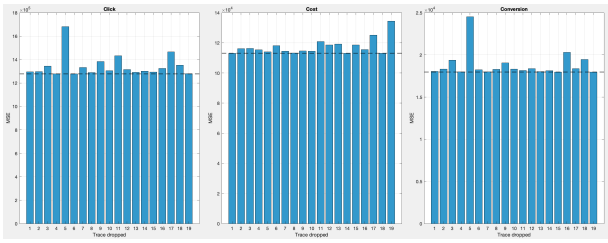



Figure 1.5: MSE computed every time a trace is dropped.

### 1.2.3. Point 2c)

Here, I needed to understand if it is possible to estimate the value of advertising $T_K$ from the analysis of the prediction error. To do so, I computed the prediction error $e(t) = x(t) - \hat{x}(t)$ for all the time instants. Then, I extracted a time window with length $W$ (that slides across all the data) and fitted a simple straight-line model: $y_j = b_0 + b_1 t_j$ with $j = 1, 2, ..., W$. The slope $b_1$ and the intercept $b_0$ are found in matrix form as (OLS solution of linear regression model):

$$\begin{bmatrix} b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} \sum t_j^2 & \sum t_j \\ \sum t_j & W \end{bmatrix}^{-1} \begin{bmatrix} \sum t_j y_j \\ \sum y_j \end{bmatrix}.$$

Eventually, I computed the coefficient of determination $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$ where $SS_{res} = \sum_{j=1}^{W}(y_j - \hat{y}_j)^2$ is the residual sum of squares, $SS_{tot} = \sum_{j=1}^{W}(y_j - \bar{y}_j)^2$ is the total sum of squares and $\bar{y} = \frac{1}{W}\sum y_j$ is the mean. The coefficient of determination tell us about the linearity of the prediction error ($R^2 = 0$ means no linearity, $R^2 = 1$ means a perfect line). Since in a certain time interval after the excitation $T_K$ the trend is approximately linear, I searched for the time window in which the behavior could be approximated as the most linear (when $R^2 \geq 0.8$ or the global maximum) and choose the starting time of the window as the excitation $T_K$. The choice of the window length W is very important: larger W gives smoother, less noisy $R^2$ but delays detection by roughly W days, smaller W reacts faster but can produce fake results on random fluctuations. After some attempts I noticed that a window length equal to 50 gives an optimal estimation for Click KPI (figure 1.6), while a window length equal to 80 gives an optimal estimation for Conversion KPI (figure 1.7).

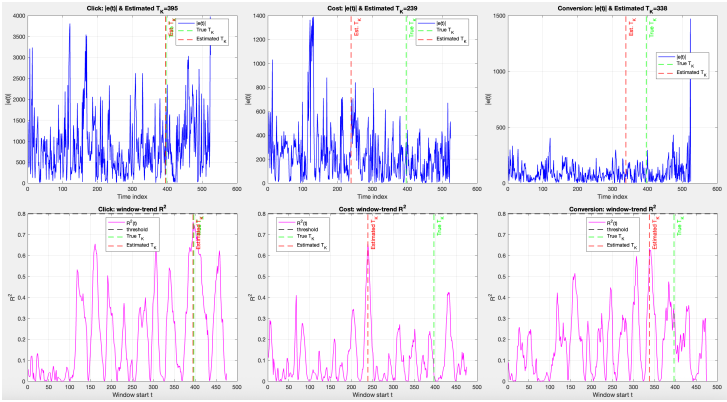



Figure 1.6: Prediction error and $R^2$ for $W = 50$.

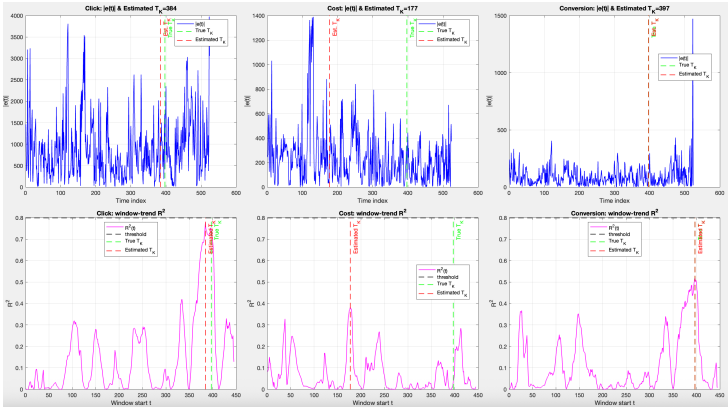



Figure 1.7: Prediction error and $R^2$ for $W = 80$.

## 1.3. Part 1 - Point 3)

This time, I needed to repeat point 2 using all KPIs at the same time (Cost and Click and Conversion).

### 1.3.1. Point 3a)

Repeating the same procedure as above (but using all the traces also of the other KPIs), I obtained the results below.

In general, for all the traces the estimation has improved (in figure 1.8 is shown the prediction trend for the trace K=20). In fact, the overall MSE has reduced by a factor 10 (look figure 1.9).

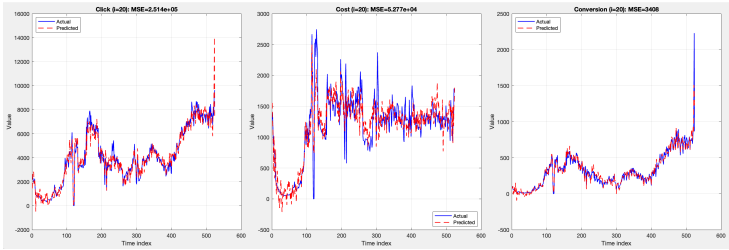



Figure 1.8: Actual and predicted values for the series K=20.

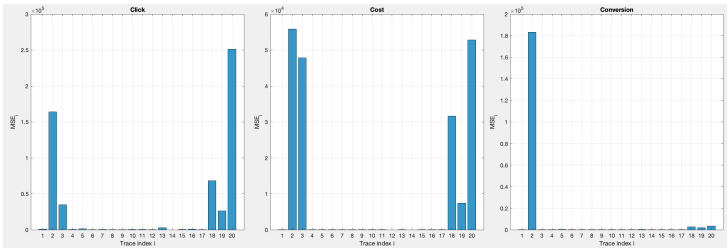



Figure 1.9: MSE computed for different traces and KPIs.

## 1.3.2. Point 3b)

As we can see in figure 1.10, also using all the KPIs at the same time, none of the traces leads to a lower MSE if dropped. We conclude that all the traces are useful to estimate the trace of interest (K=20).

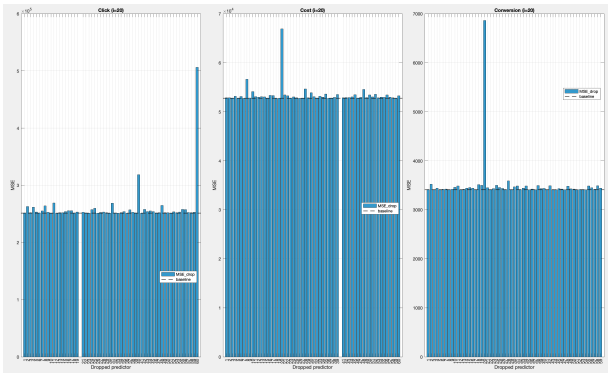



Figure 1.10: MSE computed every time a trace is dropped.

## 1.3.3. Point 3c)

Here, the estimation of the excitation time $T_K$ (using $W = 90$, figure 1.16) ha worsened for the Conversion but improved in Click and Cost.

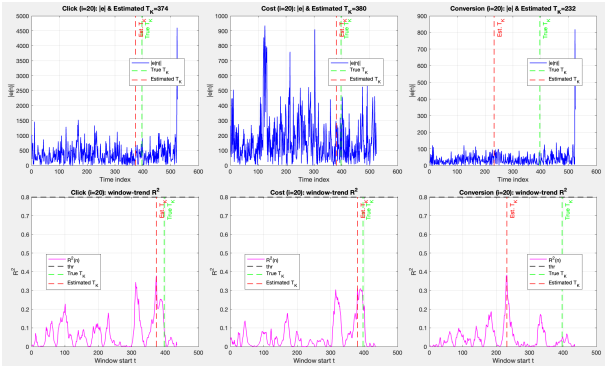



Figure 1.11: Prediction error and $R^2$ for $W = 90$.

## 1.4. Part 1 - Point 4)

This time, I needed to repeat point 3 but using memory estimators. I used a causal estimator because for a future estimation in time, I need to use only past time samples. I built a regressor vector defined as

$$\mathbf{u}(t) = \begin{bmatrix} x_{all1}(t-1) & x_{all2}(t-1) & x_{allM}(t-1) & \cdots & x_{all1}(t-2) & \cdots & x_{allM}(t-2) & \cdots & x_{allM}(t-L) \end{bmatrix}$$

where $\{all1, ..., allM\} = \{1, 2, ..., 3K\} \backslash j$ are the indexes of all traces except the one that I want to estimate, $M = 3K - 1$ and $L$ is the length of the memory estimator (number of past samples used for the estimate).

Then, I built the regressor matrix $\mathbf{U} = \begin{bmatrix} \mathbf{u}(L+1)^T \\ \mathbf{u}(L+2)^T \\ \vdots \\ \mathbf{u}(T)^T \end{bmatrix}$ and the target vector $\mathbf{y} = \begin{bmatrix} x_j(L+1) \\ x_j(L+2) \\ \vdots \\ x_j(T) \end{bmatrix}$ for the j-th trace of interest.

Finally, as before, I found $\mathbf{w}_i = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{y}$ and the estimate $\hat{y}(t) = \mathbf{u}(t)^T\mathbf{w}$

For the length L of the memory estimator, I implemented a loop in Matlab over different sizes of L (from 1 to 100) to understand which was the one the minimizes the MSE for the estimation of the 20-th trace (for all the KPIs). The results is: $L = 93$ for Click, $L = 90$ for Cost, $L = 99$ for Conversion. As a compromise among the 3 KPIs, I chose $L = 95$.

### 1.4.1. Point 4a)

The prediction result is far better than the previous points' methods (almost perfect, look figure 1.12). Also, the MSE for the estimations has become almost null (figure 1.13) for all the traces.

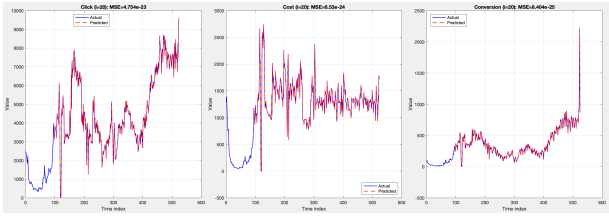


Figure 1.12: Actual and predicted values for the series K=20.

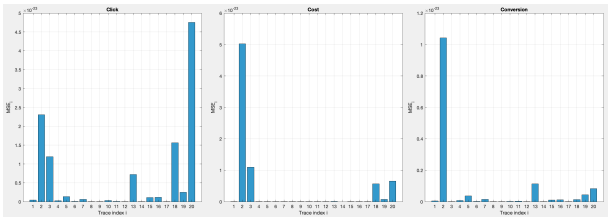


Figure 1.13: MSE computed for different traces and KPIs.

### 1.4.2. Point 4b)

Here, apart from a few cases, the most of the traces contribute for a good estimation of the 20-th trace.

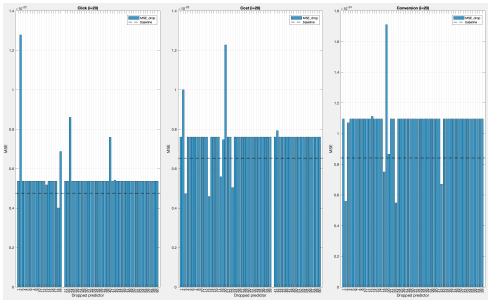


Figure 1.14: MSE computed every time a trace is dropped.

### 1.4.3. Point 4c)

The time of excitation estimation seems to be not improved a lot than before. With a window length of 10 samples $T_K$ estimated for Click and Cost is very close to the real value and for W=100 the estimation is better for Cost and Conversion.

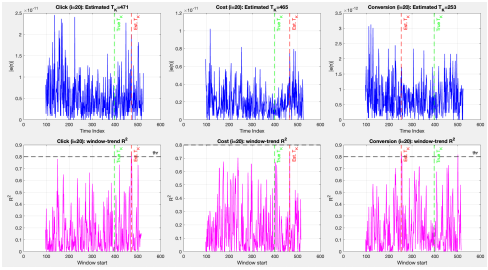


Figure 1.15: Prediction error and $R^2$ for $W = 10$.

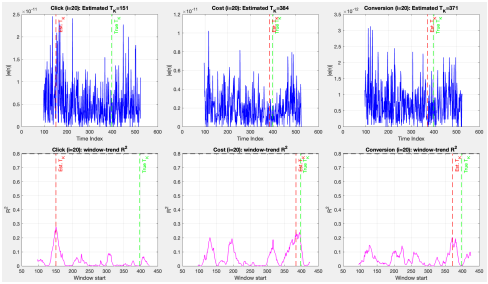


Figure 1.16: Prediction error and $R^2$ for $W = 100$.

## 1.5. Part 2 - Point 1)

For this part, we need to consider $x[n]$ characterized by a discontinuity in the statistical properties of the random process (non-stationary): I have $\alpha[n]$ ($AR(1)$ with a pole in $(\rho, 0)$) for $n \leq T_o$ and $\beta[n]$ ($AR(2)$ with poles in $\frac{\rho}{2}\exp(\pm j\phi)$) for $n > T_o$. Both AR random processes have power $\sigma_\alpha^2$ and $\sigma_\beta^2$ respectively.

So before the change point $T_o$ (AR(1)) I have $\alpha[n] + a_1^{(1)}\alpha[n-1] = v[n]$, $v[n] \sim N(0, \sigma_a^2)$.

Knowing that there is only one pole in $(\rho, 0)$, $A(z) = 1 + a_1^{(1)}z^{-1} = 1 - \rho z^{-1}$ and so $a_1^{(1)} = -\rho$. Eventually, $A(z) = 1 - \rho z^{-1}$.

After the change point $T_o$ (AR(2)), I have $\beta[n] + a_1^{(2)}\beta[n-1] + a_2^{(2)}\beta[n-2] = w[n]$, $w[n] \sim N(0, \sigma_b^2)$.

Knowing that there are two poles $p_{1,2} = \frac{\rho}{2}\exp(\pm j\phi)$, $A(z) = 1 + a_1^{(2)}z^{-1} + a_2^{(2)}z^{-2} = (1 - p_1 z^{-1})(1 - p_2 z^{-1}) = 1 - (p_1 + p_2)z^{-1} + p_1 p_2 z^{-2}$.

But $p_1 + p_2 = \frac{\rho}{2}e^{j\phi} + \frac{\rho}{2}e^{-j\phi} = \rho cos(\phi)$ and $p_1 p_2 = \frac{\rho^2}{4}$, therefore $a_1^{(2)} = -\rho cos(\phi)$ and $a_2^{(2)} = \frac{\rho^2}{4}$. Eventually, $A(z) = 1 - (\rho cos(\phi))z^{-1} + \frac{\rho^2}{4}z^{-2}$.

Then, for each AR model, I computed the prediction error as $e[n] = x[n] - \hat{x}[n|n-1]$ where the one-step prediction at time n is $\hat{x}[n|n-1] = -\sum_{i=1}^{p} a_i x[n-i]$ (p is the order

of the AR model).

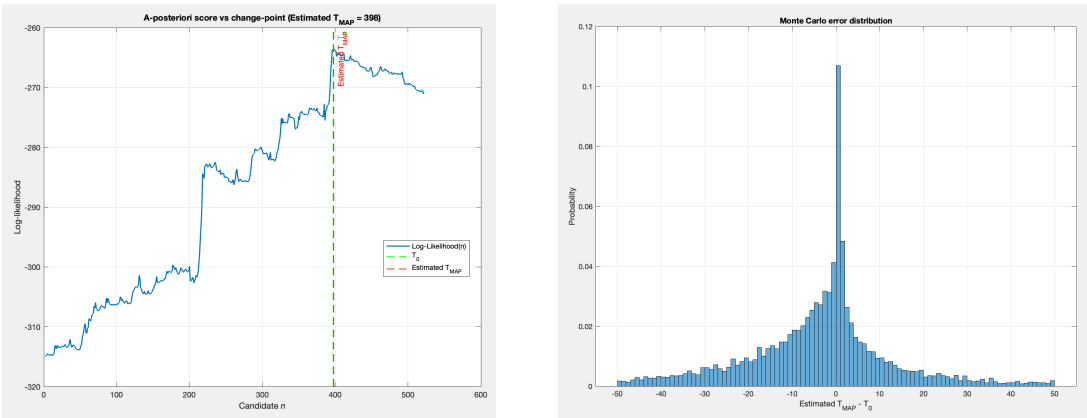
The Gaussian log-pdf for a zero-mean sequence of prediction errors $e[n]$ with variance $\sigma^2$ is given by: $log(p(e[1], ...e[N])) = -\frac{N}{2}ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N} e[n]^2$. The general Log-likelihood is $\mathcal{L}(x|\theta) = ln(p(x|\theta))$.

The total Log-likelihood is given by the sum of the two parts $\mathcal{L}_{TOT} = \mathcal{L}_\alpha + \mathcal{L}_\beta$.

The MAP estimator is the value of the time index which maximize the a-posteriori pdf (likelihood).

As we can see in figure 1.17a, the estimation of the change time is quite optimal.

Looping over 10000 Montecarlo iterations, I calculated the MSE and the bias of the estimator (Bias=-6,452 and MSE=711,829). I also computed the probability of obtaining a value of $\hat{T}_{MAP} - T_o$, the most of the computations gives a value of $\hat{T}_{MAP}$ very close to $T_o$.




(a) (b)

Figure 1.17: (a)Log-likelihood vs time index n and T estimation and (b)Probability of a value of $\hat{T}_{MAP} - T_o$.

## 1.6. Part 2 - Point 2)

Now, I needed to repeat Point 1, this time assuming that discontinuity classifier didn't know the two AR processes. So, I needed to change the way the prediction error is computed. I wrote the AR(p) in regression form as $\mathbf{y} = \mathbf{\Phi}\theta + \mathbf{e}$, where

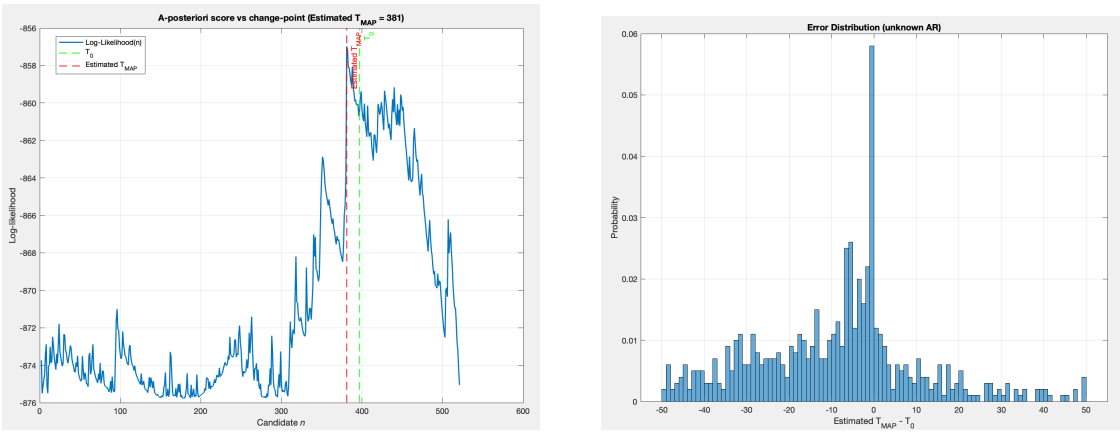
$$\mathbf{\Phi} = \begin{bmatrix} x(p) & x(p-1) & \cdots & x(1) \\ x(p+1) & x(p) & \cdots & x(2) \\ \vdots & \vdots & & \vdots \\ x(N-1) & x(N) & \cdots & x(N-p) \end{bmatrix}, \mathbf{y} = \begin{bmatrix} x(p+1) \\ x(p+2) \\ \vdots \\ x(N) \end{bmatrix}, \theta = -\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix}.$$

The Least-Squares solution leads to $\hat{\theta} = (\mathbf{\Phi^T\Phi})^{-1}\mathbf{\Phi^T y}$, recovering the AR coefficients $a_k = -\hat{\theta}_k$ where $k = 1, .., p$.

Once the coefficients are known, I can compute the prediction errors as $e = y - \Phi a$.

With unknown AR processes the estimation worsen a lot. In figure 1.18b, we see that the the error is more distributed than centered in 0 (no error).

The MSE computation gives a value of $MSE = 14095.32$ (and Bias=-64,372), much higher than before.




(a) (b)

Figure 1.18: (a)Log-likelihood vs time index n and T estimation and (b)Probability of a value of $\hat{T}_{MAP} - T_o$.

## 1.7. Part 2 - Point 3)

For this point, I needed to repeat point 19 but using a Neural Network with 1 hidden layer trained with a set of learning stages to detect the discontinuity $T_o$.

Firstly, I generated a training set using Monte Carlo simulations: I built processes as before (AR(1) before the change point and AR(2) after), but with a different random $T_o$ at every iteration and I stored their Log-likelihood into a matrix (feature matrix $S$).

Then, I shuffled the dataset (columns of matrix **S**) and I split the data into Training Set $X_{TRAIN}$(70%), Validation Set $X_{VAL}$(15%) and Test Set $X_{TEST}$(15%). I normalized the

data(zero mean and unit variance) $X = \frac{X-\mu_X}{\sigma_X}$ and scaled the time targets (between $[0, 1]$) $T_o = T_o/N$.

For the hidden layer of the Neural Network I choose the sigmoid as activation function so that at each neuron $h = \sigma(z) = \frac{1}{1+e^{-z}}$ and $z = W_1 x + b_1$(W: weights vector, b: bias vector). The final neuron (output) is $y = W_2 h + b_2$.

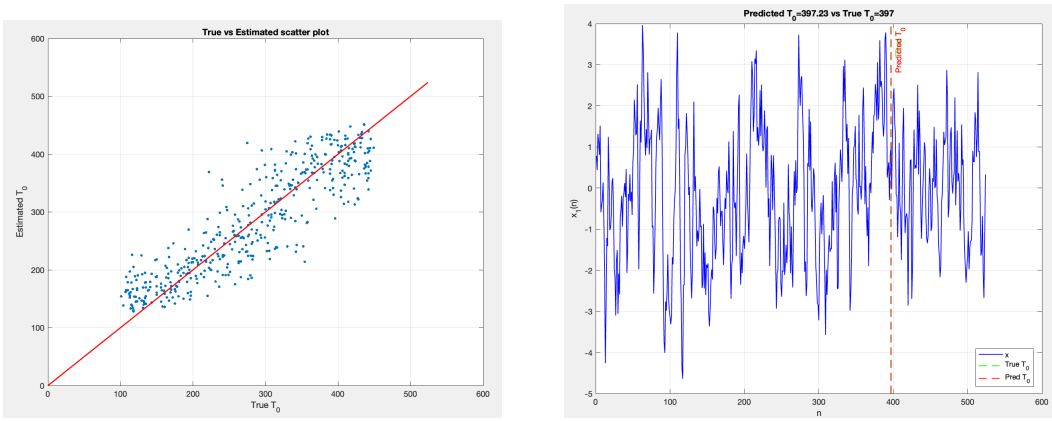
After the forward pass, I need to update the weights and biases using the gradient descent method: we move each W and b a little opposite to its gradient (backpropagation) ($W \leftarrow W - l\nabla_W L$ and $b \leftarrow b - l\nabla_b L$, where l is the learning rate). I computed the gradient using the chain rule:
$\nabla_{W2} L = \frac{2}{M}(\hat{y} - y)h^T$ , $\nabla_{b2} L = \frac{2}{M}(\hat{y} - y)$
$\nabla_{W1} L = W_2^T \frac{2}{M}(\hat{y} - y)h(1-h)X^T$ , $\nabla_{b1} L = W_2^T \frac{2}{M}(\hat{y} - y)h(1-h)$ .

Furthermore, in order to avoid overfitting (fit the noise instead of the data) I enabled early stopping with a validation step.

At every cycle, I compute the MSE of the validation set (using the updated weights). If the validation MSE stops improving for a fixed number of epochs (patience), I stop training and keep the weights that gave the lowest validation error (early stop).

For the best training, I adjusted the number of neurons in the hidden layer, the learning rate size and the number of training loops to have the minimum MSE at the exit.

Finally, I used my Test Set to see if my Network worked well. The results of the bias and the MSE on the Test is: Bias=0.082 and MSE=1977.497 . I also plotted a scatter plot of the Test series to see if the estimated change Times are similar to the true change Times (figure 1.19a), the points are located around the red line (perfect estimation) meaning that the estimate is quite good. I also did an estimation on a single series in which the time to estimate was $T_{TRUE} = 397$, as we see in figure 1.19b the estimated is quite perfect (the error is just 0.23).




(a) (b)

Figure 1.19: (a)Scatter plot of the estimates and (b)x vs time index n and T estimation.

## 1.8. Part 2 - Point 4)

I repeated point 1) looping over different values of $\rho$ (from 0.1 to 0.9 , figure 1.20). The MSE reduces as the value of $\rho$ increases. This is because the magnitudes of the poles (directly proportional to $\rho$) increase and so it is easier to detect the change because the regimes of the two AR processes become more distinct in their temporal behavior.

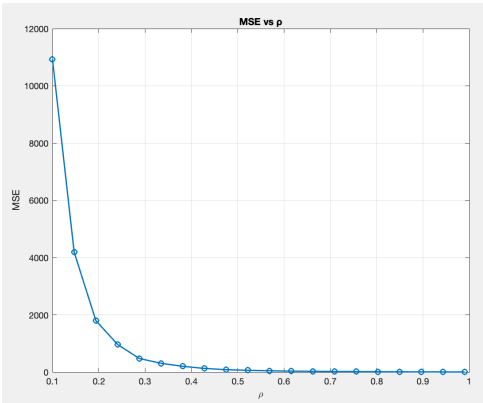
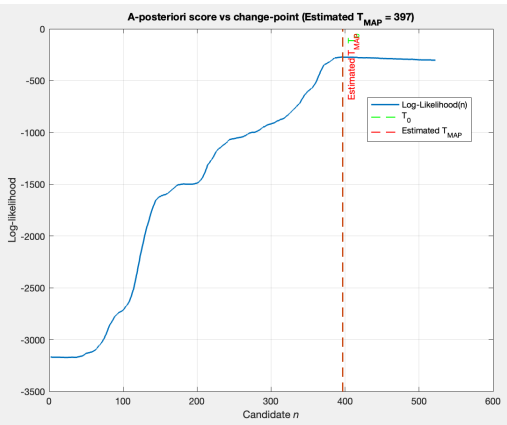
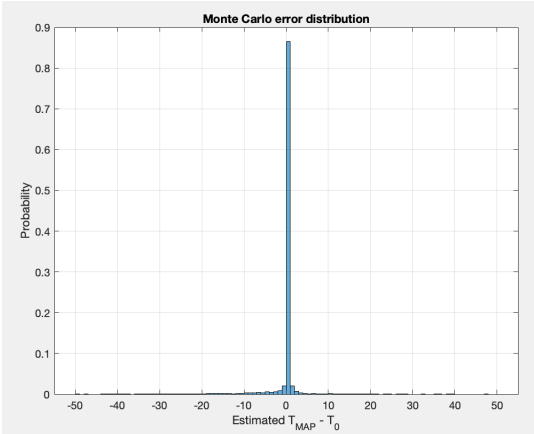



Figure 1.20: MSE vs $\rho$.

As a feedback, I computed point 1) with $\rho = 1$(best for MSE). The estimate is now optimal and the probability of having an error in the estimate equal to zero is around 85%.




(a)




(b)

Figure 1.21: (a)Log-likelihood vs time index n and T estimation and (b)Probability of a value of $\hat{T}_{MAP} - T_o$.

# 2 | Conclusions

Some of the possible improvements that could be done to solve the problem (especially for the Neural Network part) could be: increase the number of hidden layers, put the starting point for the weights in a more intelligent position, use a larger set of data.

At the end, we can see that the resolution of the homework has been successful both for the google dataset part and for the detection of non-stationary processes part.