Introduction to Signals

Juan Manuel Aragon Armas, s253163

Prof. Roberto Garello

***Abstract*—In a digital world, a signal is a sequence of values that represents the evolution in time of a physical quantity.**
Today, with computers we can process them with functions, working on the discrete domain, and then visualize the results in the time domain.
This laboratory consisted of plotting signals from a set of data using *Matlab* and studying the effect of applying some functions such as the sum, product, shift and convolution.

I. Exercise 1.1

IN this exercise a sinusoidal signal was generated by choosing as its frequency, f_0, any integer number from 1 to 10. And with a random offset phase, ϕ, between 0 and 2π radians.
Then, a rectangular pulse $P_T(t)$ was generated, using the built-in function *rectangularPulse()* with a fixed time duration of $T = 1$ second.
Finally, it was obtained the product of both signals as follows: $s(t) = P_T(t)sin(2\pi f_0 t + \phi)$.

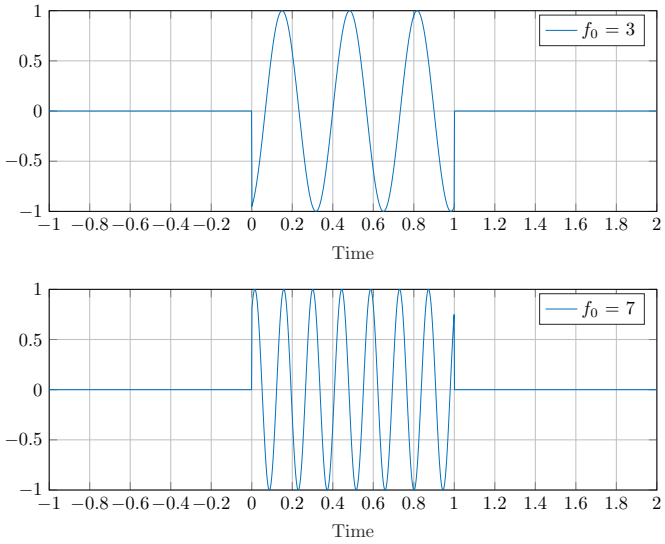

Fig. 1: $s(t)$ with $f_0 = 3$ and $f_0 = 7$.

As mentioned before, when we process signals on a computer we do it in the discrete time domain, because we are dealing with a set of values and not with continuous quantities.
That is why the first thing to do was to set up the time variable, to be an array of numbers equally spaced.
The δt chose was 0.001 units in order to have enough values to see a smooth graph.
As the function *plot()* on *Matlab* matches two consecutive points with a straight line instead of an interpolation function, by making the samples closer together the risk of having big variations got reduced.

Some examples of the possible results are shown in *Fig.1*.
It can be noticed that the amplitude of $s(t)$ continued to be between 1 and -1, because the sinusoidal samples were multiplied with only two possible values (1 or 0).
The domain instead, became [0,1]. Considering that the rectangular pulse function is zero out of that time interval, the picture clearly shows that $s(t)$ became zero where at least one of the two functions was zero.
On the other hand, the two plots started with a different initial amplitude and slope, because of the random offset phase.

II. Exercise 1.2

THe second exercise introduced the concept of convolution between signals, in particular, between two rectangular pulses. One of them whose parameters were adjustable, such as its amplitude (A), time duration (T) and time delay (D).
These signals were represented on a 10s time interval, with samples at every $\delta t = 0.001$s, starting from zero.

The output signal was obtained by applying the the built-in function *conv()*, and normalizing it by the δt mentioned before.
Because of *matlab* calculates the discrete convolution as a sum of products, by increasing δt it

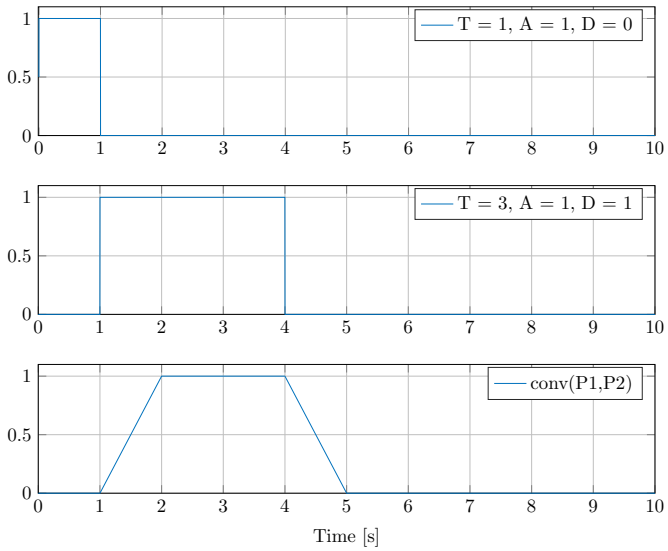

Fig. 2: Convolution of rectangular pulses with different time duration T

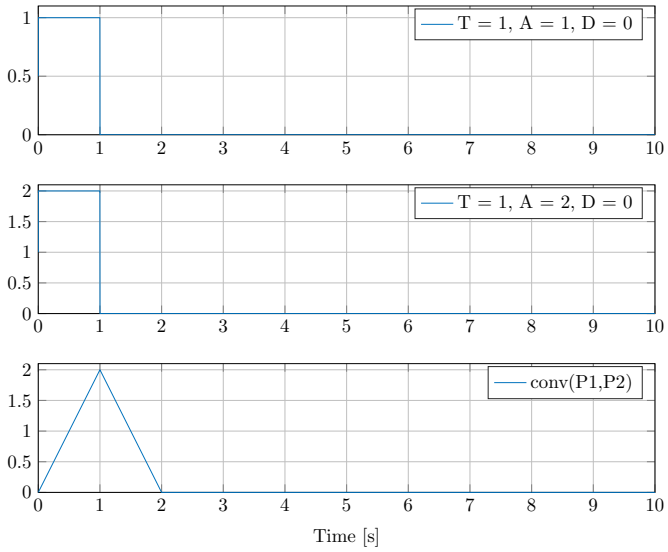

Fig. 3: Convolution of rectangular pulses with same time duration T

also increases the number of values in the signal vector, and so modifying the result. Therefore, it was necessary to normalize the result according to the length of the signal.

Fig.2, shows an example of the output obtained. In general can be done a geometrical reasoning to verify that the output is correct. Considering T1 and T2, as the time duration of the rectangular pulses, P1 and P2. It can be proved that the output of the convolution between P1 and P2 is a trapezoid shaped signal, whose long base equals the sum of T1 and T2. While its short base equals the absolute difference between T1 and T2.

In a similar way, it can be deduced that the convolution between two identical rectangular pulses would be a triangular pulse, as shown in *Fig.3*.

Another interesting observation is that the introduction of a delay in time by any of the input pulses does not modify the shape of the output signal.

III. Exercise 1.3

FOR the third exercise were defined a hundred sinusoidal signals, with frequencies going from 1 to 100 Hz respectively. Each signal with a randomly generated offset phase going from 0 to 2π radians.

From that set of signals were obtained 6 output signals. The first one, $s_{10}(t)$, corresponded to the signal having frequency of 10 Hz. From its graph (*Fig.4*) can be seen that it had a uniform oscillation due to the presence of a single frequency. It also showed 10 peaks per every 1 second interval, as expected.

The second plot, $s_{20}(t)$, corresponded to the signal having frequency of 20 Hz. It was as regular as the previous one but with the double amount of peaks in the same time interval.

In the same way the third signal, $s_{100}(t)$, made with a frequency of 100 Hz, was observed a uniform oscillation on time, and traced a higher quantity of peaks per time unit.

On the other hand, the rest of the signals were no longer uniform because they consisted of sums of signals at different frequencies. It can also be noticed that the amount of peaks per second was no more easy to determine as in the previous signals.

For example, the signal $s_a(t)$, was the result of the sum of ten sinusoidal signals having frequencies from 1 to 10 respectively. From its graph, *Fig.7*, it was not even possible to determine which frequencies were present.

At *Fig.8*, can be seen a noisy graph, made by the sum of all the hundred signals defined previously. In comparison to $s_a(t)$, it showed higher variations on time. And also its amplitude laid on a wider range of values.

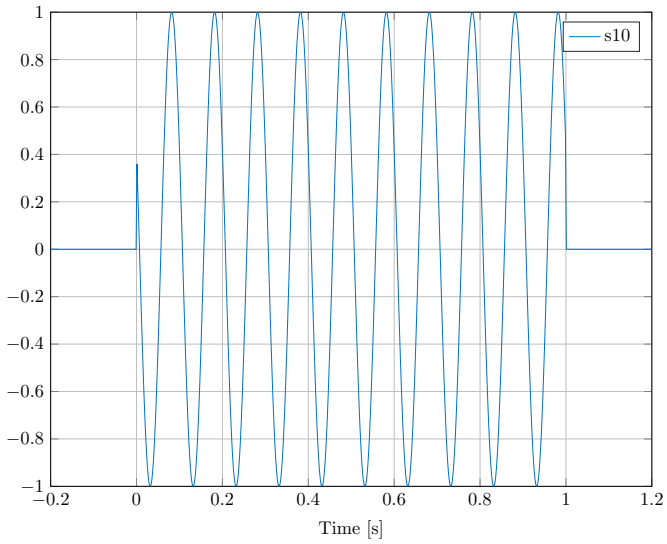

Fig. 4: signal $s_{10}(t)$

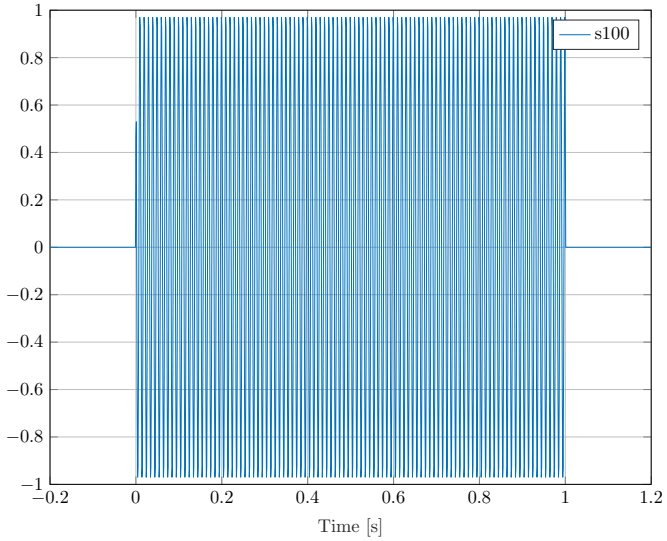

Fig. 6: signal $s_{100}(t)$

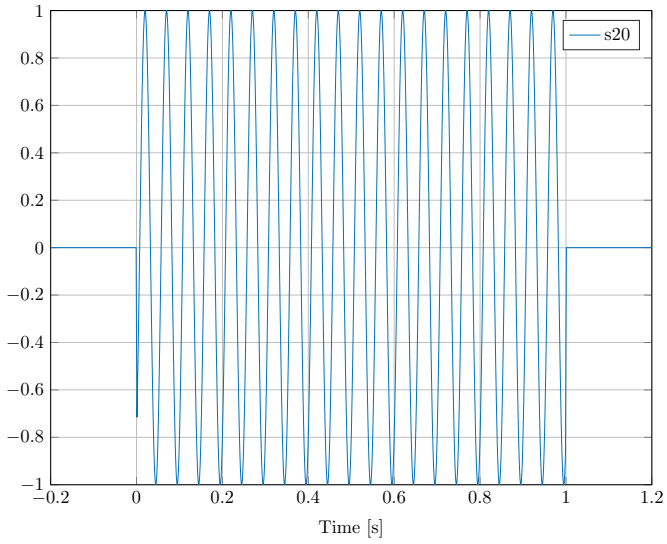

Fig. 5: signal $s_{20}(t)$

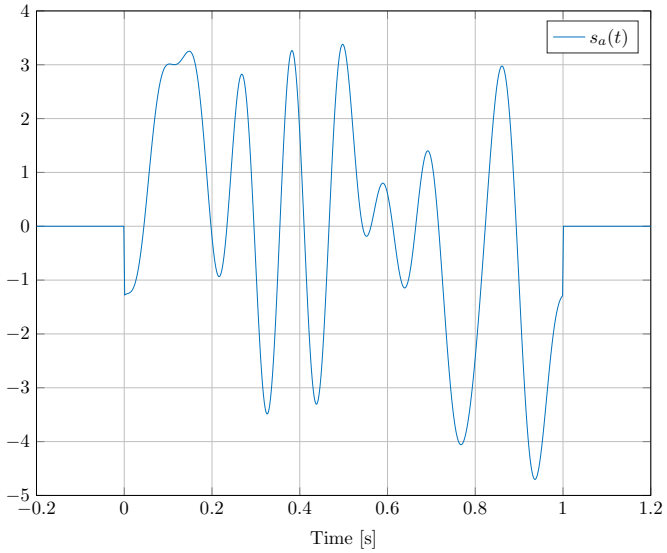

Fig. 7: signal $s_a(t)$

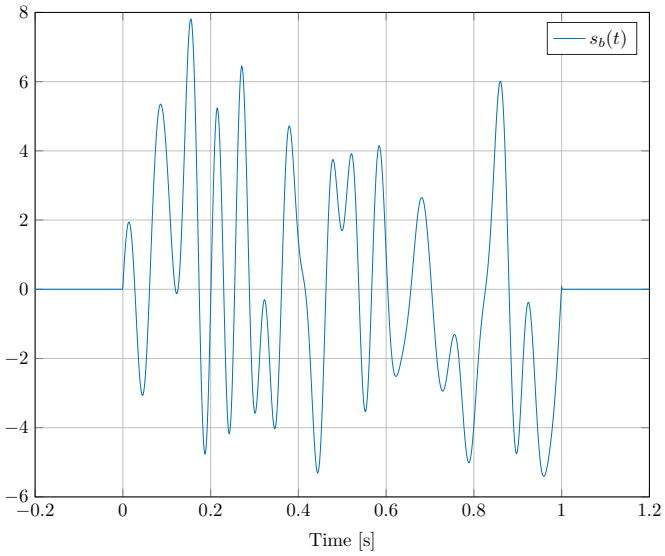

Fig. 8: signal $s_b(t)$

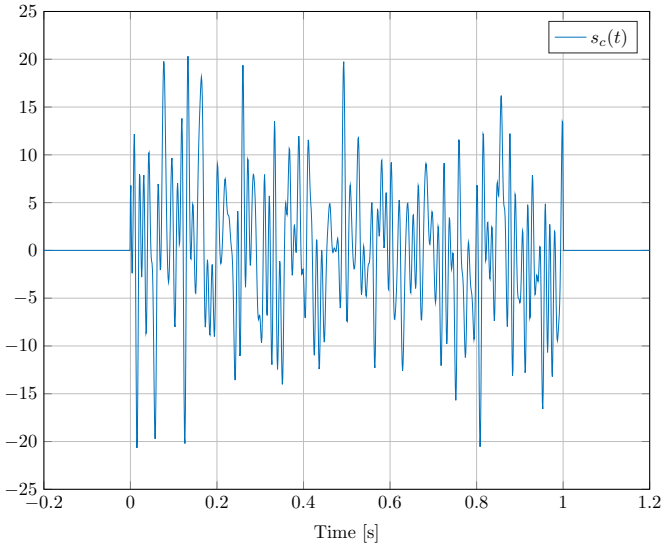

Fig. 9: signal $s_c(t)$

Energy, Discrete Fourier Transform

Juan Manuel Aragon Armas, s253163

Prof. Roberto Garello

***Abstract*—When working with signals must of the analysis have to be done in the frequency domain.**
Given a continuous signal expressed on the time domain, the *Fourier Transform* is a mathematical function that allow us to visualize the same signal at every frequency component.
For digital signals, the ones we process in our computers, should be applied a discrete version of that function, such as the *FFT* or *Fast-Fourier-Transform*.
In this laboratory some signals were studied using the *FFT*, analysing their energy and understanding some sampling properties.

I. Exercise 1.4

A rectangular pulse, $y(t) = AP_T(t)$, was defined on *matlab*. Manipulating its amplitude, *A* and time duration, *T*. The goal was to compute the *FFT* and to calculate its energy.

There are some important terms to clarify before explaining the code and its results.
The *Observation Time*, *To*, refers to the time range on which the samples of the signal were defined. In this case, being ten times the time duration of the pulse. While the *Sampling Time*, *Ts*, refers to the time separation between two successive samples. In this case, being 0.002s.
It can also be deduced that the total amount of samples to work with were *N = To/Ts*.

On the *matlab* code, the first thing to do was to define the time vector. Consisting of *N* values going from 0 to *To-Ts* seconds.
With this vector, a digital version of the signal was defined, *y(t)*, being this the signal of interest to apply the *FFT*.
In adition to the *matlab* library function *fft()*, two more operations were necessary.
Those were, a normalization, consisting of a multiplication by the factor *Ts*. And the application of *fftshift()*, which rearranged the vector obtained from the *fft()*, swapping the left and right halves in order to get the zero-frequency component at the center of the array.
In *Fig.1* can be observed the results of this part. The second graph of the figure shows a limited frequency axis going from -2.5 to 2.5 Hz. The only scope to show only those frequencies was to see with more detail the shape of the function, knowing that it tends to zero when going far from the origin. In the *matlab* code that is why a *cutData()* function was implemented.

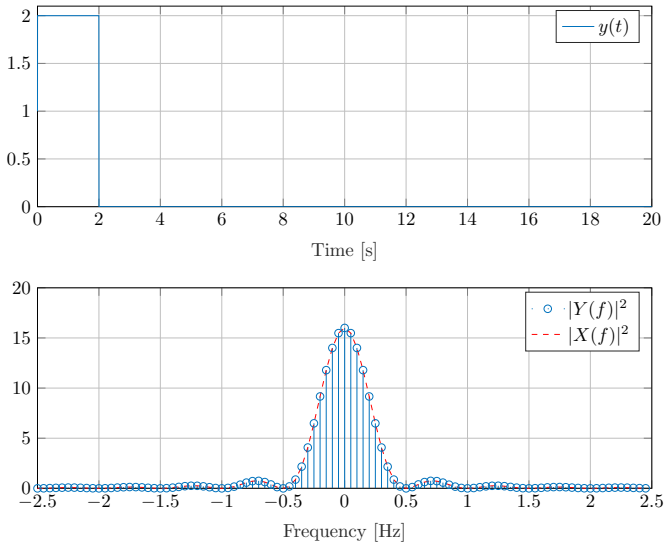

Fig. 1: *FFT* applied to a rectangular pulse

As expected, the *FFT* graph seems to have the shape of $|X(f)|^2 = |ATsinc(fT)|^2$. It can also be seen that it has its zeros at every multiple of *1/T*, except at the origin, where it has its maximum equal to A^2T^2.

In case of the continuous signal *X(f)* can be found that its energy is A^2T. By using the formula:

$$E(x) = \int_{-\infty}^{\infty} |X(f)|^2 df. \tag{1}$$

On the next part of the laboratory, the energy of the discrete signal was calculated as the area under the curve of $|Y(f)|^2$ using the function *trapz()*.

From the results showed on *matlab* can be noticed that the energy of the discrete signal corresponds to the 99.95% of the exact value.

A more detailed study about this was made on the third part of the laboratory.
Calculating multiple times the energy of the signal considering only the frequencies from 0 to n/T for $1 \leq |n| \leq 100$. Plotting the results in two separate graphs. *Fig.2* containing the first 10 lobes and *Fig.3* containing lobes from 10 to 100.

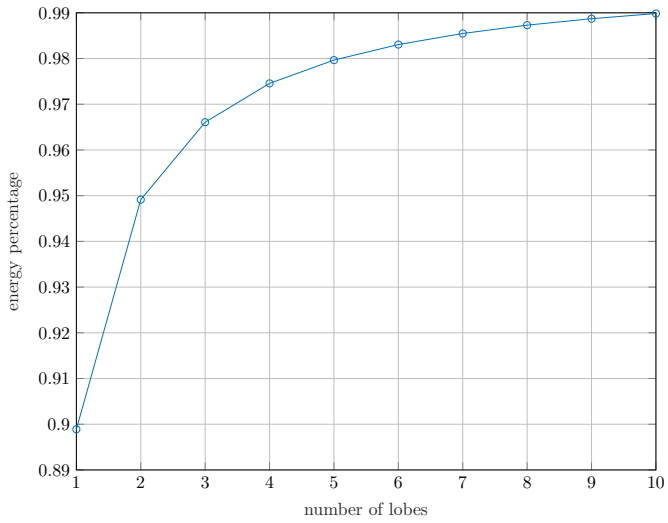

Fig. 2: Energy percentage lobes 1-10

As mentioned before, the energy of the digital signal goes beyond the 99% of the expected value, meaning that the *y(t)* was a good approximation for *x(t)*, or in other words, that the values *Ts* and *To* were right for obtaining the digital signal.

From the first graph, it can also be seen that the mayor contribution happens in the first lobes by reaching nearly the 95% just after the second lobe.
The graphs show only the first 100 lobes but can be said that by adding more lobes the percentage will continue to raise, every time in a smaller quantity, until it reaches the 99.95% discussed above.

Finally, in order to improve the approximation, can be chose a lower Sampling Time, to have samples closer together. Instead defining a higher *To*, will not change the obtained results, even if in both cases the number of samples, *N*, is increased.

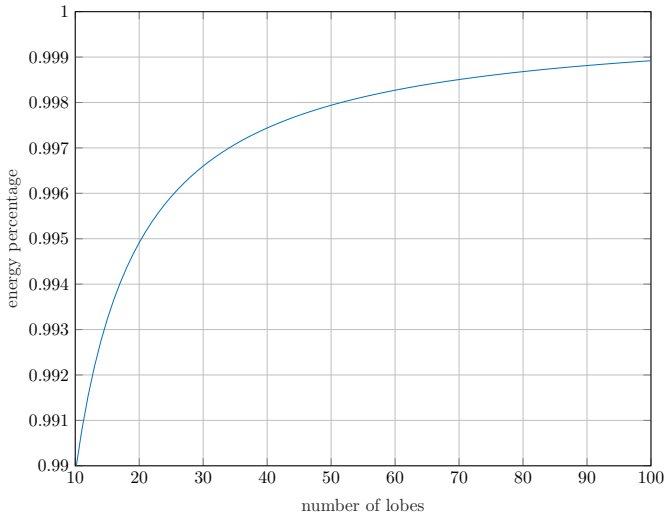

Fig. 3: Energy percentage lobes 10-100

This is because when increasing *To* nothing new is being discovered about the pulse, only more zeros are being defined on the graph of *y(t)*, meanwhile by lowering the Sampling Time a more detailed shape of the pulse is obtained, and then a better *FFT* can be made.

II. Exercise 1.5

The second exercise was more focused on the sampling theory, that was slightly discussed on the first exercise.
It is necessary to understand what the sampling frequency is, and what the *Nyquist Theorem* says about it.

The Sampling Frequency, *Fs*, is nothing less than *1/Ts*. And refers to the number of samples per unit time that the digital signal will be made of.
According to the *Nyquist Theorem*, when a periodic signal is being sampled the sampling frequency must be at least twice the highest frequency component of the signal. When this is not satisfied, aliasing is generated. Meaning that the behaviour of the sampled signal is not as expected.

For the first part of the exercise it was generated a digital signal, made with the sum of three sinusoidal functions, having frequencies of 10, 20 and 100 Hz respectively. Each sinusoidal had also its own random offset phase, uniformly distributed between 0 and 2π.

The scope of this exercise was to see how the spectrum changes when the signal is being sampled at different values of *Fs*.

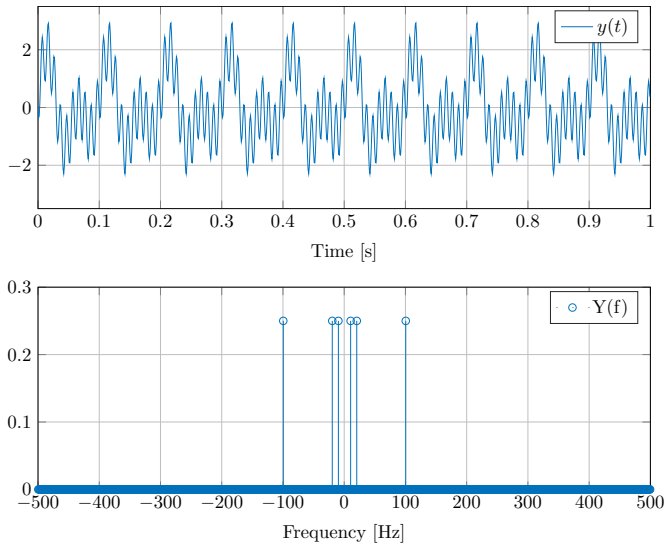

Fig. 4: Samplig at frequency 1000 Hz

Fig.4 shows the results of applying *Fs* = 1000 Hz. As can be noticed the *Fs* satisfies the *Nyquist Theorem*. Being 100 Hz the highest frequency component of the signal and 200 Hz the minimum value acceptable for *Fs* in order to avoid aliasing.
The second graph of the figure, displays the same signal on the frequency domain an it clearly shows Kronecker deltas at $\pm 10, \pm 20$ and ± 100 Hz. As the *Fourier Transform* of a sin wave are two deltas at $\pm f_0$.

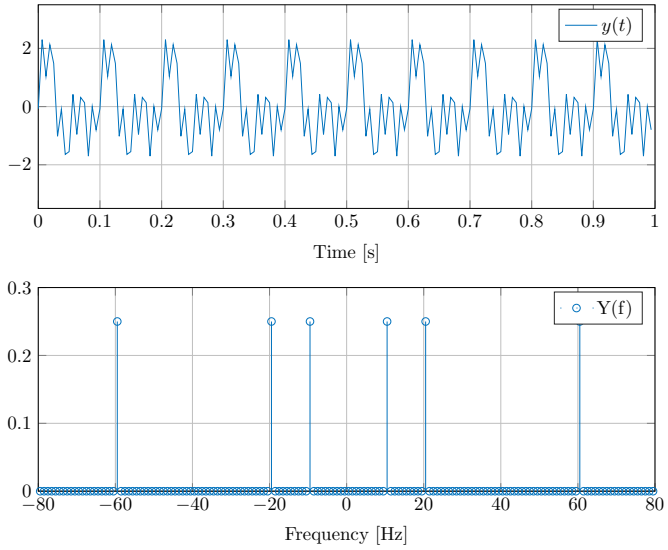

Fig. 5: Sampling at frequency 160 Hz

Fig.5 instead shows a different result, as the signal was sampled with a frequency smaller than one suggested by *Nyquist Theorem*. Can be seen that the shape of the sampled signal is different than the previous one. As a result of using less samples per unit of time, and the presence of aliasing.
By analysing its frequency components can be identified components at 10, 20 and 60 Hz. The third component was artificially generated by the aliasing phenomenon. In other words the 100Hz component of the neighbour Fourier copies of the sampled signal felt on the 60Hz component.

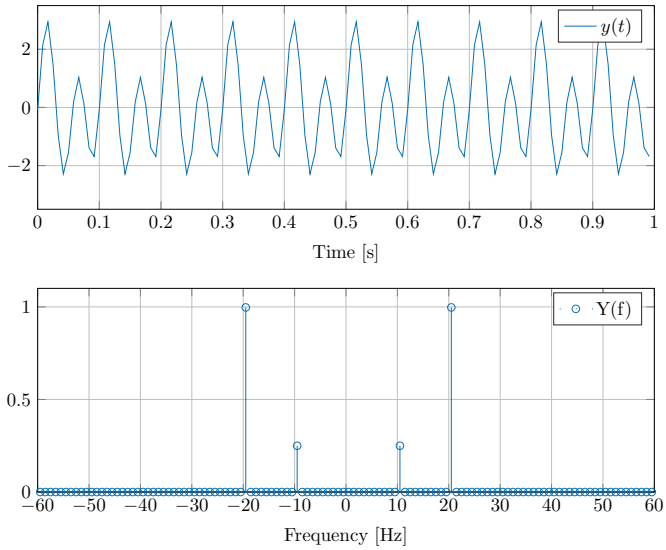

Fig. 6: Samplig at frequency 120 Hz

Finally by reducing more the *Fs*, to 120 Hz. The effects of the aliasing were more evident. On *Fig.6*, the first graph shows a much more rough wave signal apparently made with only two frequency components, at 10 and 20 Hz.
Although at *Fs* = 160 Hz, where aliasing was evident, the Kroneker deltas had all the same height. At *Fs* = 120 Hz, the heights are different. This is because, in this case the 100Hz component of the neighbour Fourier copies of the sampled signal matches with the component at 20Hz.
The 10Hz component remains invariant, with a value of 0.25.

Signal Correlation

Juan Manuel Aragon Armas, s253163

Prof. Giuseppe Rizzelli

***Abstract*—An important measure used in the study of signals is the correlation. It tells us how similar two signals are, taking in consideration the presence of delays in time. In this laboratory were studied the signal's auto and cross-correlation, analysing its properties and experimenting with real-life applications.**

I. Exercise 1.1

THE first exercise consisted of generating a periodic signal, *x[n]*, made with 50 samples and a time period of $T = 5s$, as it is shown in *Fig.1*.

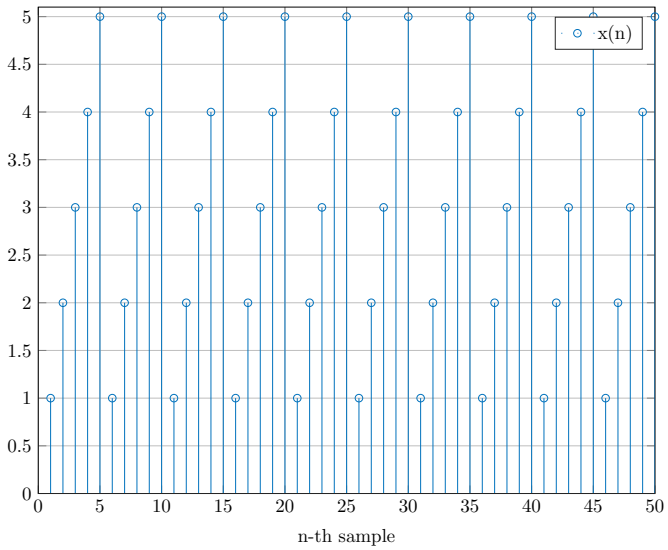

Fig. 1: Periodic digital signal with 50 samples

The goal was to calculate the auto-correlation function using three different methods, and to evaluate their similarities and differences.
For the first one, the iterative method, was implemented a function on *matlab* called *myCorr()*, that received by parameter the sequence *x[n]*. Then using two nested for loops it was calculated the auto-correlation array summing the products of the original signal with a shifted version of itself.
The second method consisted of using the *matlab* built-in function *xcorr()*, again receiving by parameter the signal *x[n]*.
And finally, the third method was about the built-in function *conv()*. In this case the parameters were the signal *x[n]* and a flipped version of itself obtained with *flip(x)*.

All the results are shown in *Fig.2* . As expected they are all identical, meaning that all the three methods are valid ways for calculating the auto-correlation of a digital signal.

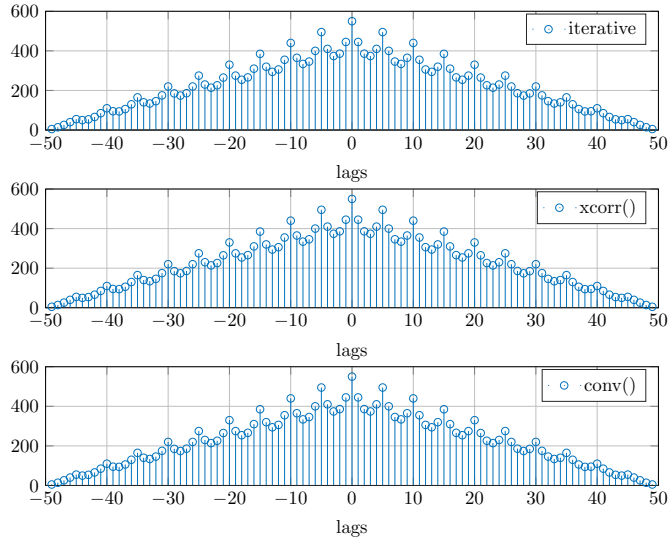

Fig. 2: Auto-correlation of x[n]

By analysing the results, can be noticed that they all have a symmetric triangular shape with a global maximum at the origin. That maximum corresponds to the value of the signal's energy, meaning that in this case the energy of *x*, *E(x), had a magnitude of 550*.
It can also be noticed that there are some others local maximums along the slopes, that lay always above coordinates multiples of T = 5. This happens because when shifting the replica of the signal, at every T, some periods get out of range while the rest of the signal matches perfectly with the original

x[n].
This type of analysis demonstrates that all the properties of the auto-correlation are satisfied.

II. Exercise 1.2

As on the previous exercise, a new periodic signal, *y[n]*, was generated. Made with 80 samples an a time period of $T = 10s$, as shown in *Fig.3*.

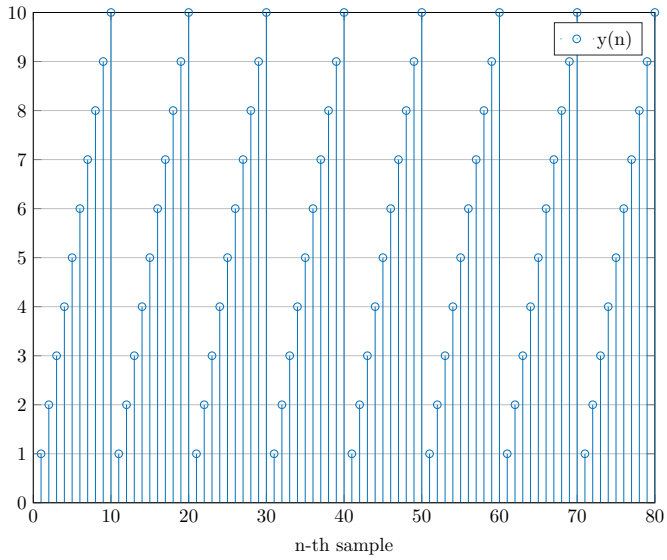

Fig. 3: Periodic discrete signal with 80 samples

In this case the goal was to calculate the cross-correlation between *x[n]* and *y[n]* using three different methods.
The first one consisted of implementing an iterative procedure, writing a function called *myXCorr()*. It is quite similar to the *myCorr()* function mentioned before, as it uses two nested loops to shift and sum the products of the two signals.
The second method involved the built-in function *xcorr()*, receiving *x* and *y* as parameters.
And finally using *conv()*, but in this case with a flipped version of *y[n]* done with *flip(y)*.
In *Fig.4* are displayed all the three output signals.

This time the output graphs have a trapezoidal shape with peaks along the slopes and on the upper base.
It can be noticed that the graphs are no more symmetric at the origin, as it was expected.
Watching the peaks, they all lay above an abscissa

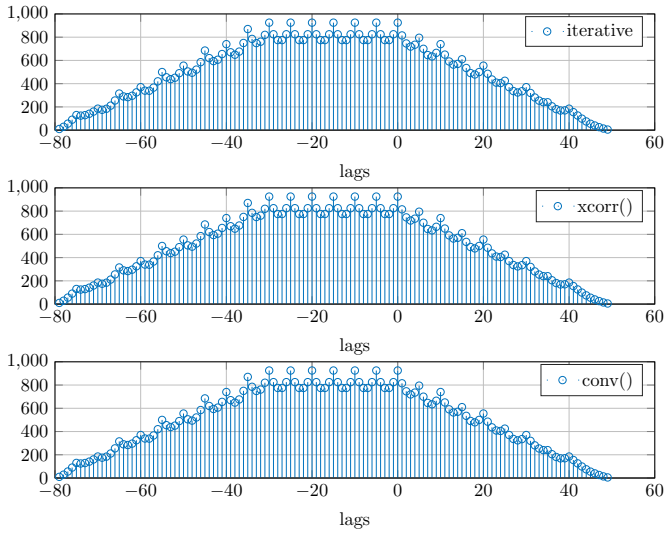

Fig. 4: Cross-correlation of x[n] and y[n]

multiple of T = 5. This is due to the fact that the signals have a higher similarity when the first 5 values of a period on the signal *x*, matches with the first 5 values of a period on the signal *y*, as they are almost the same. This happens at every 5 shifts. Unfortunately not all the periods matches due to the different lengths signals have.
Also, when some of the periods get out of range their correlation decreases and that is why the peaks on the slope are lower than the ones on the upper base.

III. Exercise 2.1

FOR this exercise, the goal was to determine the phase shift between two sinusoidal waves at the same frequency of 1 Hz, using again the correlation and its properties.

The first step was to generate those digital signals on *matlab*, using the same notation of the previous assignments. Sampling time, *Ts*, equal to 0.1 ms and observation time, *To*, equal to 10 s[1].

The only difference between both the signals was the offset phase, ϕ_i and ϕ_p respectively. Each one was randomly generated between 0 and 2π using *rand()*.

[1]Used To = 10 in order to see a less dynamic graph, however in *matlab* it can be reproduced the results showed on the slides by changing To = 100.

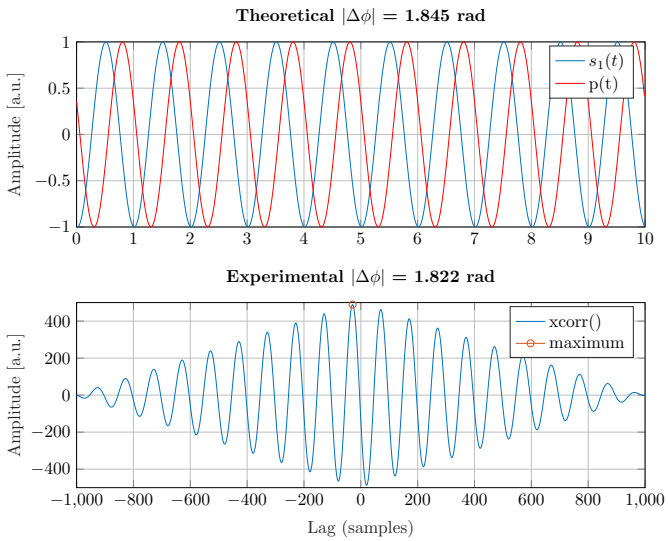

Fig. 5: Phase shift using To = 10s

As seen in the lectures, any correlation graph shows a global maximum where the best match of the signals is found. Giving a slight idea on how to determine the presence of delays, if both of the signals were supposed to be identical, by looking at the position of the global maximum.
In *Fig.5.* can be seen two graphs. The first one shows the two digital signals having each of them a different offset phase. At the top of the graph is specified the theoretical $\Delta\phi$, referring to the absolute value of the difference between ϕ_i and ϕ_p.

On the second graph instead, is shown the output obtained from the *xcorr()* function applied to both of the signals.
It has a maximum near the origin marked with a red circle. The abscissa of that maximum was then used to calculate the offset phase, in the *matlab* code expressed as *ExperimentalPhi*, and it's value can be found at the top of the graph.
It can also be noticed how precise is the phase shift function by watching the *AbsoluteError-Phi* and *RelativeError-Phi* displayed on the Command Window.

IV. Exercise 2.2

THE second part of the exercise was very similar to the first one. Now substituting the signal $s_1(t)$ with $s_a(t)$, the result of summing N signals, with N being 1, 10, 100 and 1000.
Again the goal was to determine the phase shift between $s_1(t)$ and $p(t)$ by computing the cross-correlation between $s_a(t)$ and $p(t)$, and then to compare the results with the theoretical value.

At the beginning a set of 1000 digital signals were defined on *matlab*. Each of them with its unique frequency value equal to their own index on the set.
Also for each signal an offset random phase was generated as in the previous part.
For the cross-correlation, it was used the built-in function *xcorr()*. And the rest of the procedure was almost the same. The global maximum and its abscissa were determined in order to identify the phase angle, called on *matlab* as *ExperimentalPhi*. This result was then compared with the *TheoreticalPhi*, establishing the absolute and relative errors.

It can be seen on the *matlab* code that by increasing the value of N the relative error also tends to increase but in lower scale. That is because the signal becomes more noisy by adding more signals at higher frequencies. Difficulting the process of finding the exact phase shift.

In *Fig.6.* and *Fig.7.* are shown some results obtained from the *matlab* code were it can clearly be noticed this phenomenon by comparing the theoretical and experimental $\Delta\phi$.

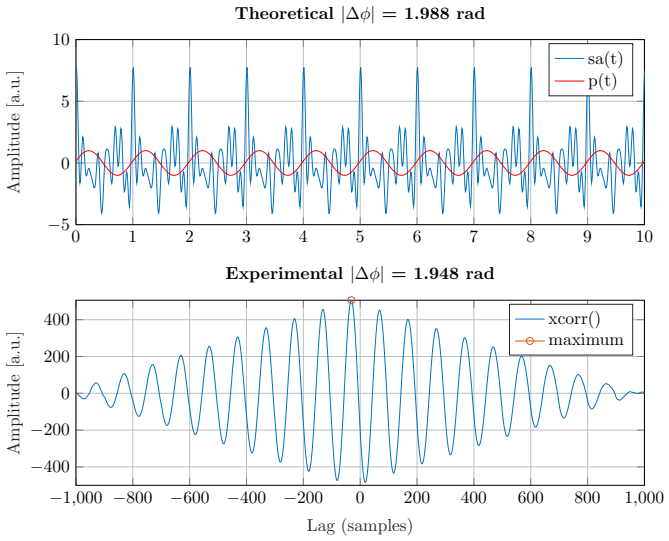

Fig. 6: Phase shift using To = 10s and N = 10.

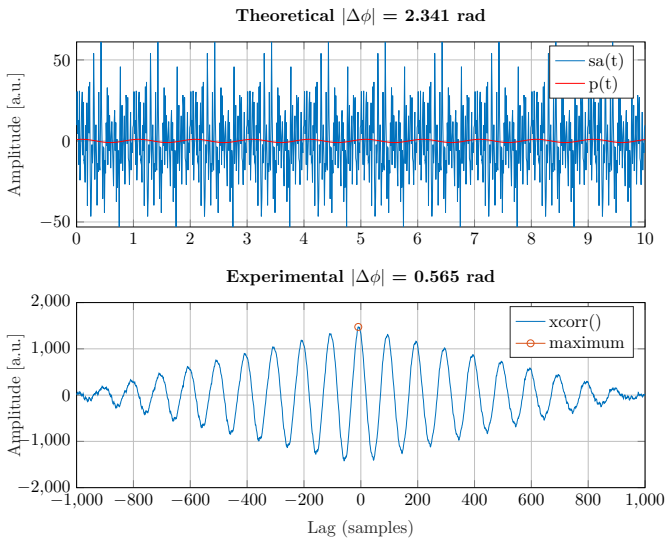

Fig. 7: Phase shift using To = 10s and N = 1000

V. Exercise 3.1

In this exercise cross-correlation was implemented once more, to determine the phase shift of a digital signal, now affected by a white noise.

First of all, a digital signal, x[n], was generated as a sequence of random bits by using the function *randi()*.
For the white noise, *N[n]*, it was generated another sequence of values having a zero mean and a specific standard deviation.

Then a third signal was generated from the sum of the previous ones, *y[n] = N[n] + x[n]*.
Fig.8 shows an example of the signals obtained at this part. As it was expected, by using a low standard deviation from the white noise, the graphs still share big similarities.
On the *matlab* code by increasing the standard deviation, the differences between y and x became be more evident.

Continuing with the exercise, the signals *xShift* and *yShift* were obtained by applying *circshift()* with a random delay to the signals *x* and *y* respectively.

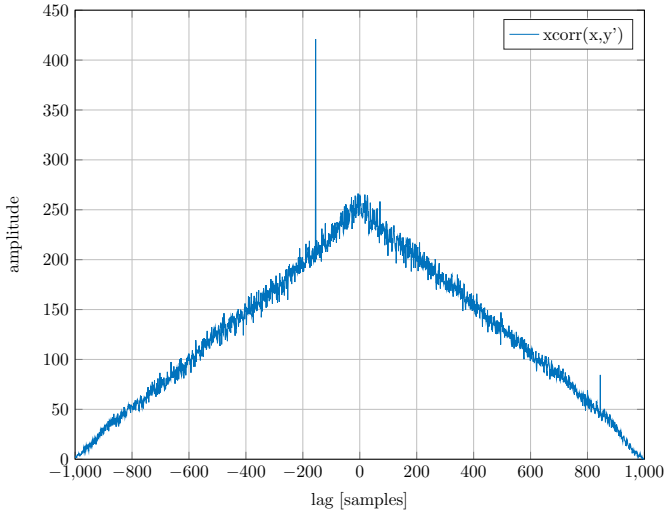

Fig. 9: Cross-correlation between x[n] and y'[n].

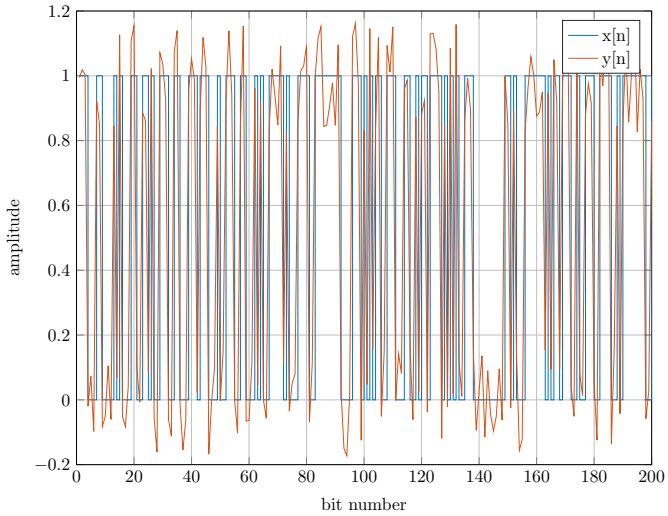

Fig. 8: White noise with std = 0.1

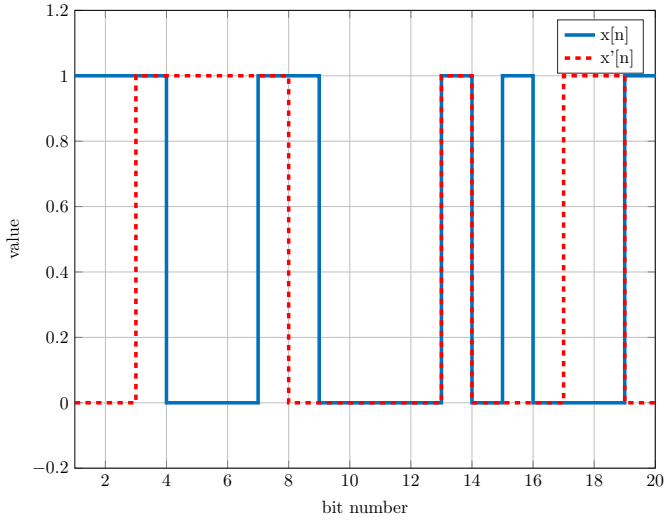

Fig. 10: Shifted version of x[n]

Once the *xcorr()* was applied to *x* and *yShift* a graph, like the one shown in *Fig.9*, was obtained. This graph was then divided by half, and determined the position of the global maximum. The reason of this division was to identify the sign of the new corrective delay.
The value instead was determined by reading the abscissa of the maximum.

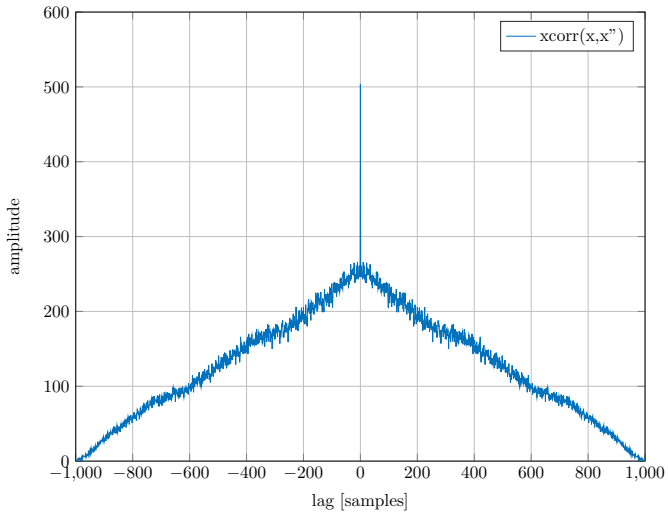

Fig. 11: Cross-correlation between x[n] and x"[n].

With this value, a new signal, *xShift2*, was generated. As seen in *Fig.12*, this signal is identical to *x*, meaning that the original delay was determined correctly.

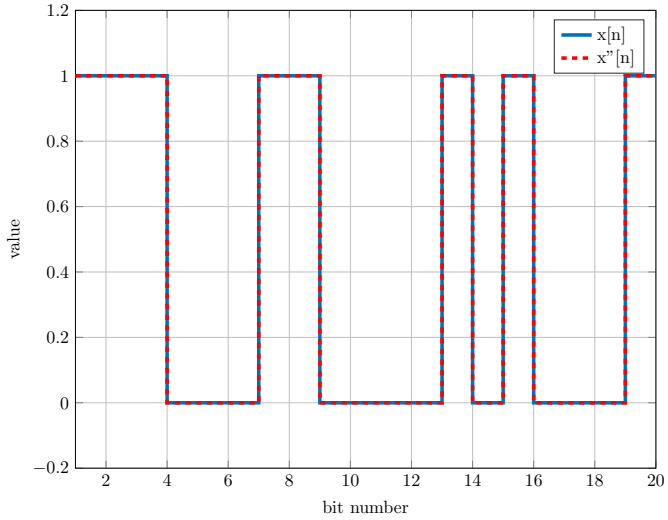

Fig. 12: Compensated version for x[n].

Finally the cross-correlation was once more applied to *x* and *xShift2*, to verify one more time that the two signals were identical by having the maximum at the origin, as it can be observed in *Fig.11*.
This whole procedure was then executed 50 times for different values of standard deviations. In order to study how it affected the result of the cross-correlation.
For each iteration, after calculating the *xShift2*, the *Bit Error Rate* was determined, by making the absolute difference of *x* and *xShift2*. *Fig.13* shows the value of how many times the calculated *BER* was not zero at every standard deviation.
It can be seen how for low values of the standard deviation it is possible to obtain *BER = 0*, instead for high values it seems the opposite.

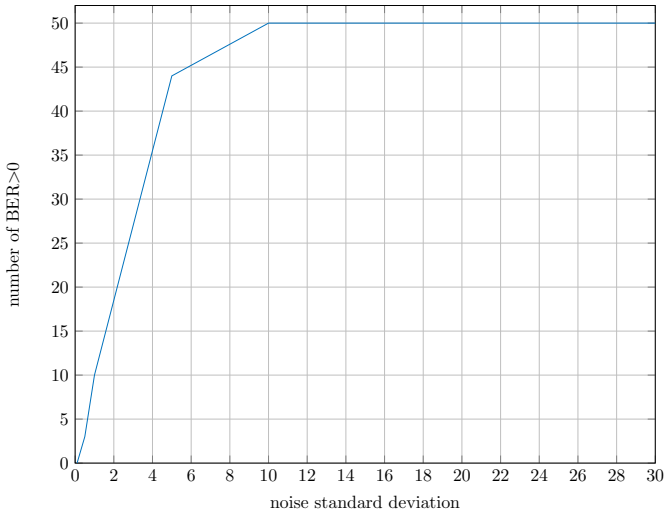

Fig. 13: Bit-Error-Rate for different values of STD.

Spending two additional words on the topic, at *Fig.14*. and *Fig.15* can clearly be seen how the standard deviation affects the process of finding the right shift delay.
In the first figure, it is quite impossible to notice any difference between the two graphs even if the upper one is a noisy version of the one at the bottom.
What happens is that the noise added to the signal has a low standard deviation which means that the real values differ a little from the original ones.
And the both maximums have the same abscissa that is then used to calculate the shift delay.
On the other hand, the second figure shows the

case on which the standard deviation of the noise is higher, and consequently, their maximum points does not correspond to the same abscissa.
This is why on the *BER*'s graph for higher deviations it is quite impossible to make a right guess of the shift delay.

By increasing the observation time, To, the number of samples shown in the cross-correlation graph will be higher, meaning that the global maximum should be identified with a better precision. But still in that case, the presence high STD will bring troubles for finding the shift delay.

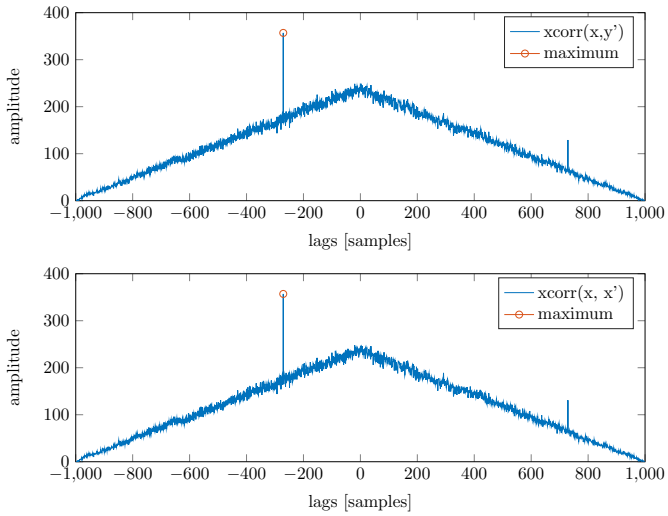

Fig. 14: *xcorr()* at STD = 0.1

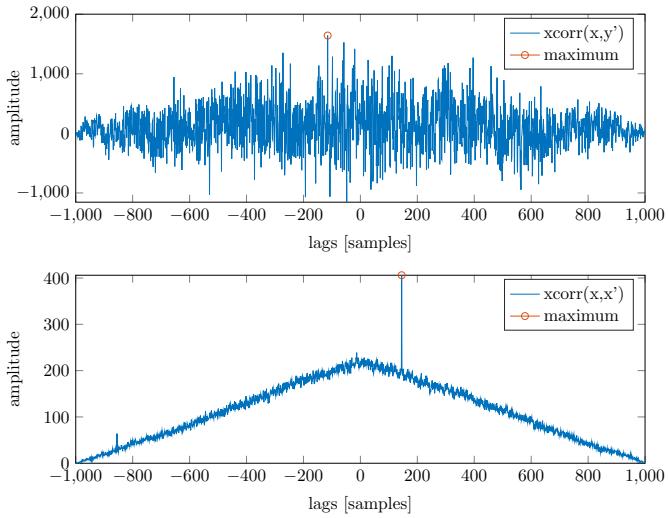

Fig. 15: *xcorr()* at STD = 30

VI. Exercise 3.2

THE second part of the exercise consisted of replicating the signal *x[n]* multiple times and to analyse how this replicas improve the *BER*'s graph for higher values of STD.
The exercise was supposed to be done with 5 replicas of *x[n]*, but in order to give a better explanation, here are shown the results of 10 replicas.

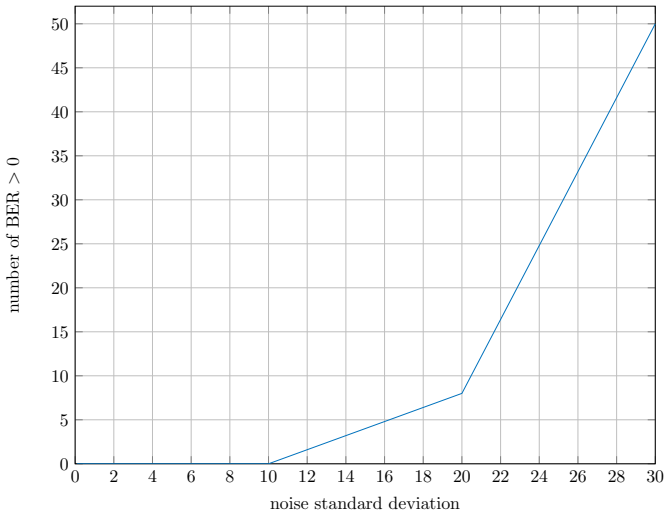

Fig. 16: Bit-Error-Rate for x[n] replicated 10 times.

In *Fig.16.* is shown the *BER*'s graph. It can clearly be seen how the curve remains low from the beginning and starts rising at very high values of STD. This means that the presence of more replicas improves the process of finding the correct phase delay from the cross-correlation in presence of white noise.
In *Fig.17.* can be understood the reason. As it can be seen on the second graph, that shows the cross-correlation between two clean signals, there appears ten peaks on both the sides of the graph instead of only one as showed before. This peaks are related to the 10 replicas that were introduced. And also the global maximum is higher than before, because a higher number of samples matched perfectly on the correlation.
So, even in presence of white noise with high values of STD, the global maximum tends to remain high, as showed on the first graph.

VII. Exercise 4.2

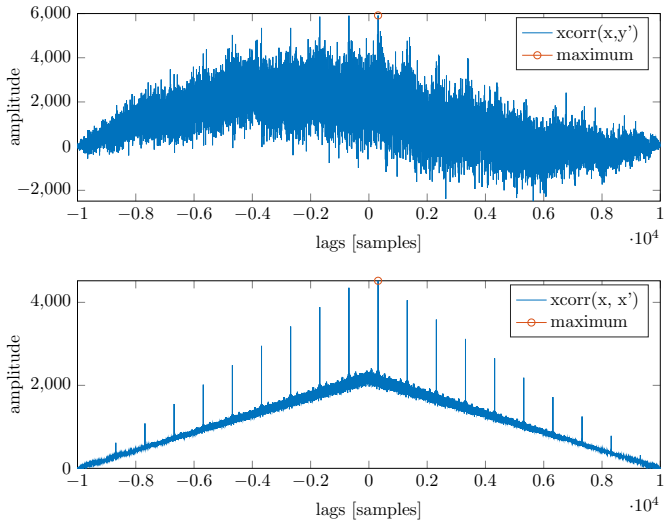

Fig. 17: *xcorr()* with STD = 20 and *x[n]* replicated 10 times

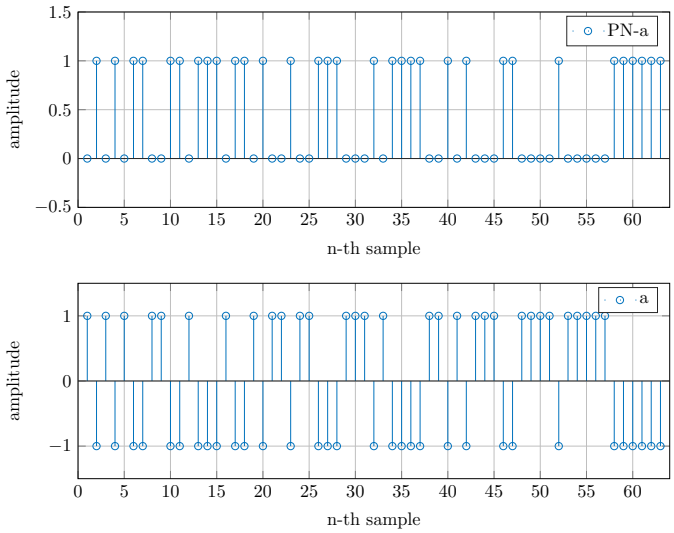

Fig. 18: m-sequence generated from a pseudo-noise sequence

FINLAY, the last exercise was more oriented on a real-life application of signal correlation. The generation of m-sequences, preferred sequences and gold codes. Used in telecommunications in order to spread multiple signals by CD-MA, giving to each signal a pseudo noise sequence that allow the signals share the same medium, time and bandwidth without interfering with each other.

The first part of the exercise consisted on generating a pseudo-noise sequence using the *LFSR* technique. In this case from a polynomial characteristic of degree $n = 6$. $p(x) = x^6 + x + 1$. Initializing each bit with the state '1'.

On *matlab* it was implemented the function *myPNgenerator()*, that receiving by parameter the degree, the polynomial characteristic and an initialization vector it produced the pn-sequence. With a for loop it shifted each state of the coefficients and calculated the *XOR* of the corresponding variables.

In *Fig.18* it is shown the obtained pn-sequence and its associated m-sequence. It can clearly be seen that the first graph has thirty two '1' corresponding to 2^{n-1}. And the rest of the bits are all zeros. Which proves the balance property.
For the run property, there are 16 runs of length 1, which is half of the total runs (32).
There are also eight runs (a quarter from the total) of length 2.
Four runs are of length 3, two runs of length 4, one run of length 5 and one of length 6. Meaning that also the run property is satisfied.

For the correlation property, the *myPxcorr()* function was implemented. It received by parameter two m-sequences and returned its normalized circular cross-correlation.

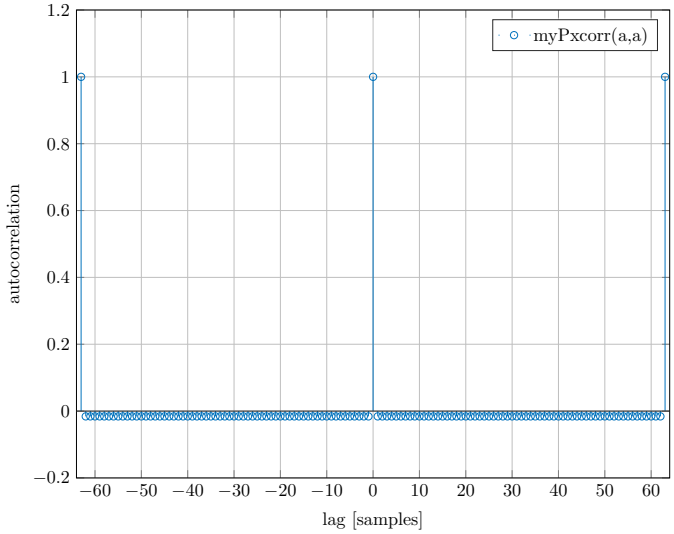

Fig. 19: auto-correlation of an m-sequence

In *Fig.19* can be seen that the auto-correlation

shows only two possible values. A '1' at the origin and at both extremes. And nearly '-0.016' in between. This proves that the correlation property is valid because the only two values expected were '1' and '$-1/2^{n-1}$' as in this the case.

Continuing with the exercise, the next step consisted of getting a decimated version of *pn-a*. In this case the decimal factor chosen was q = 5. And from the *Command Window* it can be seen that all three necessary conditions were satisfied.
These conditions were validated by the implemented method *validateConditions()* that receiving as parameters q and the degree of the polynomial. It first checked that the degree was odd or $mod(n, 4) = 2$. Then being $k = log_2(q - 1)$, it checked that the *gcd* between n and k was either 1, if n was odd, or 2 if $mod(n, 4) = 2$.

At this point the cross-correlation was applied to the original m-sequence and the one obtained from the decimated pn-sequence.

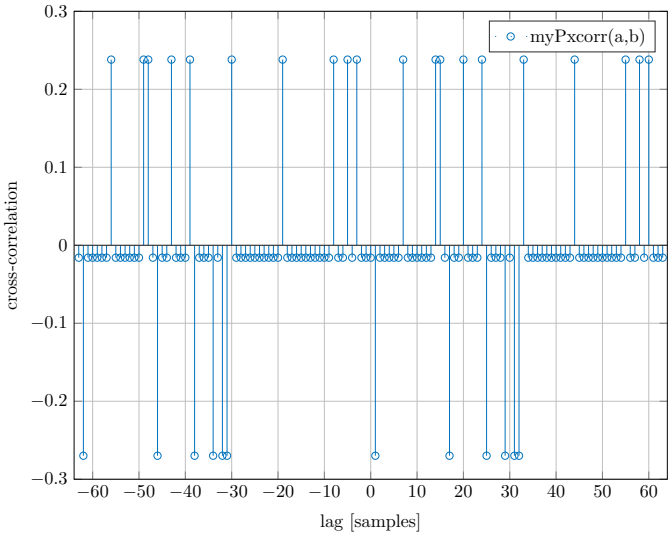

Fig. 20: cross-correlation between two m-sequences

In *Fig.20* can be seen that the output is a three-valued graph which means that both the m-sequences are what is known as *preferred pairs*. The last part consisted on generating all possible gold codes from the preferred pair obtained before. This was implemented on a for loop that shifted *pn-b* and calculated the *XOR* between *pn-a* and *pn-b* converting the result into a m-sequence and storing it on a matrix.

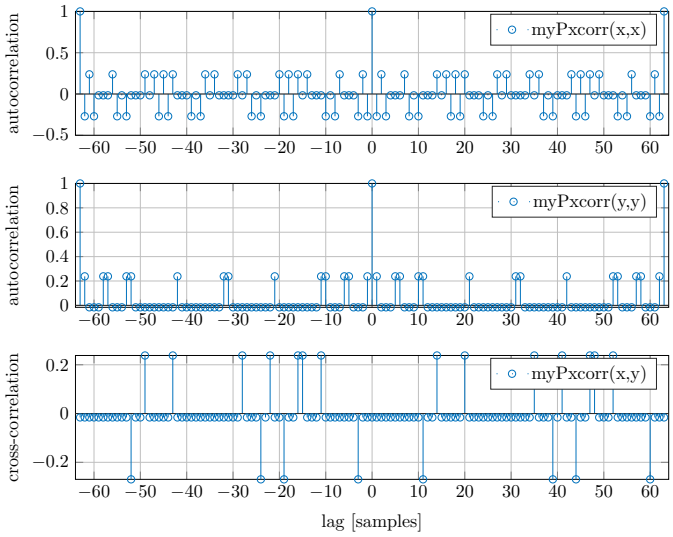

Fig. 21: auto and cross-correlation of two gold codes

In *Fig.21* can be seen the result of applying the *myPxcorr()* to two randomly selected m-sequences, x and y, from the matrix of 2^{n+1} gold codes.
As can be seen, the auto-correlation of each of the selected gold codes is not two valued, as expected. And lastly, their cross-correlation is three valued, meaning that indeed they are gold codes.

Spectral Analysis

Juan Manuel Aragon Armas, s253163

Prof. Daniel Gaetano Riviello

***Abstract*—Spectral analysis refers to the process of decomposing signals in all their frequency components, in order to study their amplitude, power and other signal properties that are more evident on the frequency domain. In this laboratory, using an implementation of the *DFT*, it was obtained the periodogram of a signal, which means the Fourier Transform of its auto-correlation. Then, it was used to compare the quality of the power spectrum estimation obtained from *Bartlett* and *Welch* methods.**

I. Exercise 1.1

THE goal of the first exercise was to compare the *sinc()* function with a periodic version of itself, obtained from the modulus of the complex *DTFT* of a rectangular pulse having duration L.
And also, to see how by increasing the duration of the pulse, more lobes seem to match with the standard *sinc()* function.

The first step was to define a vector of frequencies from -1 to 1 with 1000 points using *linspace()*.
Then with a for-loop the DTFT of a rectangular function was computed many times:

$$X(e^{j2\pi f}) = \frac{sin(\pi f L)}{sin(\pi f)} e^{-j\pi f(L-1)} \qquad (1)$$

On each iteration the value of L was replaced with integers numbers from 3 to 7.
In *Fig.1* are shown the graphs of $|X(e^{j2\pi f})|$ having L equal to 3 and 7.
It can be noticed how the value of the global maximum increase directly proportional to L while the period of both signals, is the same in this case, T = 1.

The second step consisted of calculating the function $Z(e^{j2\pi f})$, corresponding to the modulus part of the complex function $X(e^{j2\pi f})$, using the same frequency vector mentioned before.
In *Fig.2* are shown two different graphs of the function $Z(e^{j2\pi f})$.

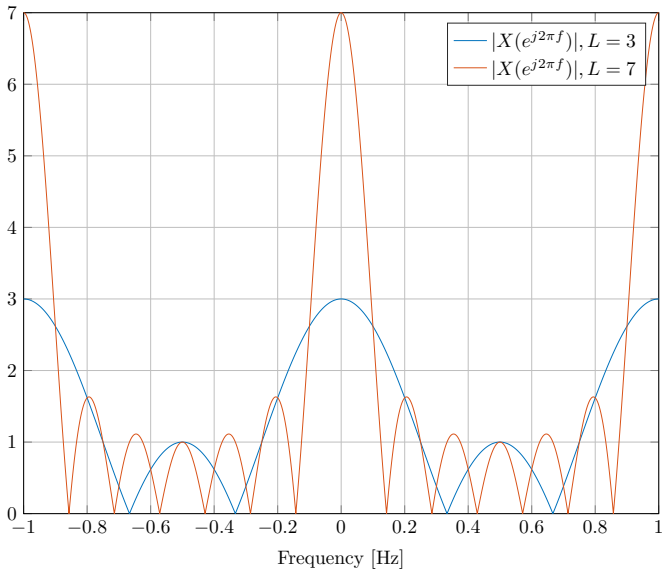

Fig. 1: Absolute values of the DTFT of rectangular pulses having duration L.

As it can be seen the first one, having L as an odd value, shows a period of T = 1, while the second one, having L as an even value shows a period of T = 2.

The last step consisted of comparing the function $Z(e^{j2\pi f})$ with the *sinc()* function: $LSinc(\pi f L)$.
From *Fig.3* can be seen that for high values of L a very good approximation is obtained around the origin.
It can also be seen that because of the *sinc()* function tends towards zero when f goes to infinity, the approximation gets worse when the periodic replicas of $Z(e^{j2\pi f})$ are reached.

II. Exercise 1.2

THE second exercise consisted of the implementation of the function *myDFT()*, that computes the Discrete Fourier Transform of a signal with an iterative procedure. Receiving by

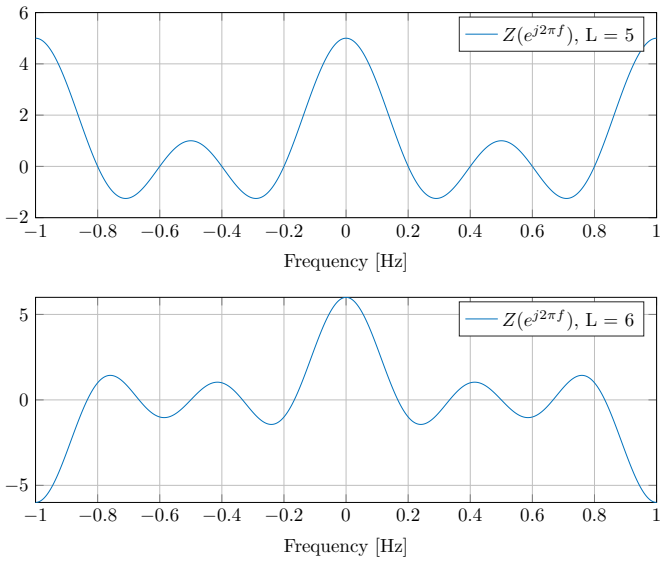

Fig. 2: $Z(e^{j2\pi f})$ for odd and even values of L

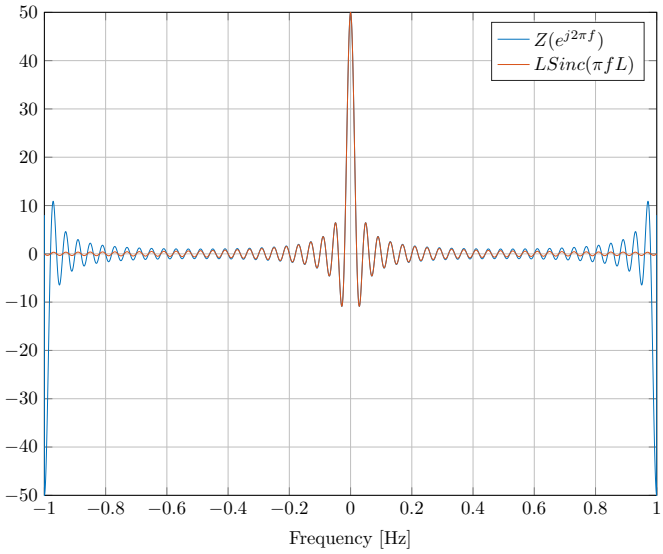

Fig. 3: Approximation of $Z(e^{j2\pi f})$ by $LSinc(\pi fL)$ at L = 50

parameter the signal vector, *x[n]*, and its length, *N*.

The first step was to define the frequency and time vectors, consisting of N consecutive integers values starting from zero.
Then the expression $X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi n\frac{k}{N}}$ was implemented by using a for-loop on which at each iteration the signal x[n] was multiplied with its own complex coefficient vector.
Finally the vector was rearranged by swapping the two halves of the resulting X[k] which was then returned.

In order to check the proper functioning of the implemented method, a signal x[n] was simulated to be the sum of two sinusoidal waves with frequencies 10 and 20 Hz. Then the DFT was calculated using the *fft()* and *myDFT()* functions, satisfying all the sampling requirements.
As expected, in *Fig.4* can be verified that both graphs show the same results. Consisting of 4 Kronecker deltas, at $\pm f_{10}$ and $\pm f_{20}$.

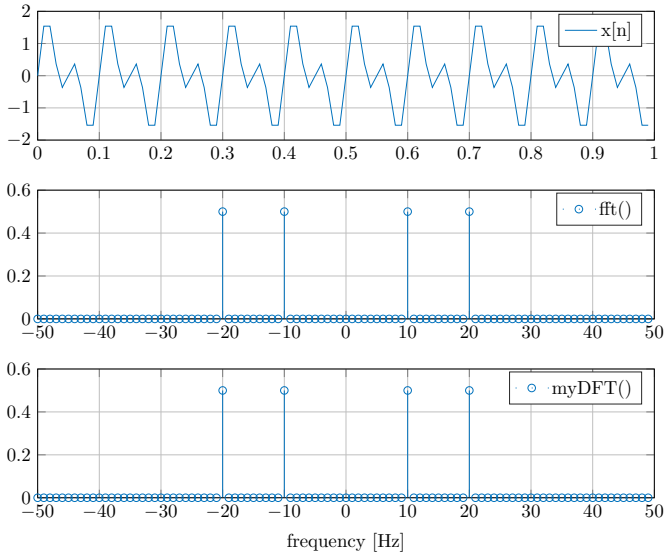

Fig. 4: Comparison between *myDFT()* and *fft()*

III. Exercise 1.3

FOR this exercise a triangular sequence, *x[n]*, was defined using the *matlab* built-in function *triangularPulse()* with a duration of T = 15.
Initially it consisted of N = 15 samples, with observation time, To = 15 s, and sampling frequency, fs = 1 Hz.

The goal of the exercise was to compute the *DFT*, using the *fft()* function, in order to obtain the signal *X(k)*. And to compare it with the known *DTFT* for triangular pulses:

$$X(e^{j2\pi f}) = \frac{sin^2(\pi fM)}{Msin^2(\pi f)} \tag{2}$$

Then to repeat the same process but increasing the observation time by adding the necessary amount of zeros to the signal without modifying the parameters of the triangular pulse.

In *Fig.5* is shown the graph of $|X(e^{j2\pi f})|$. As mentioned before, it represents the *DTFT* of a triangular pulse having duration of 15 s.
It can be seen that the graph has its zeros at every 2/T, as it was expected.

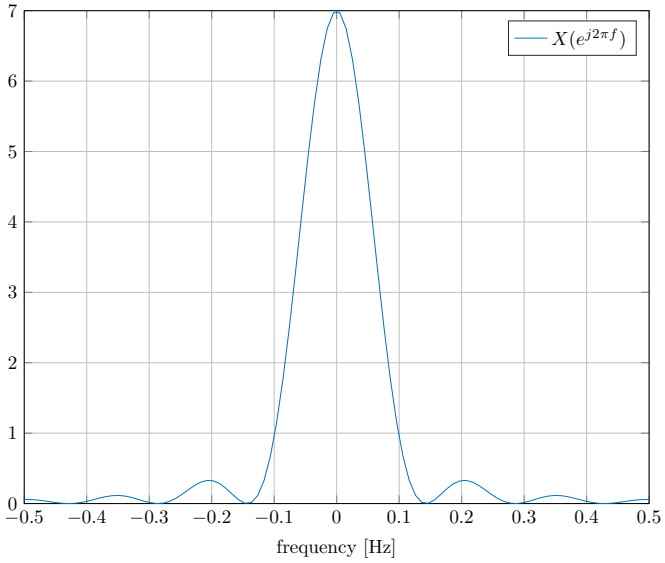

Fig. 5: *DTFT* of a triangular pulse of duration 15.

On the other hand, in *Fig.6* are displayed the sequences *x[n]* and *X[k]*, both made with the same amount of samples, N = 15. It can also be verified that the signal *X[k]* is a sampled version of $|X(e^{j2\pi f})|$.

The same process was then repeated in order to have N = 64 and 128 total samples. In *Fig.7* and *Fig.8* are shown their results. As can be noticed the effect of increasing the amount of samples N, by the zero padding method, is a more detailed *DFT* that corresponds to the sampled version of the *DTFT*.

IV. Exercise 1.4

THE goal of this exercise was to verify that the zero padding does not return ideal results when the original signal is not time limited, as the periodic sinusoidal waves.

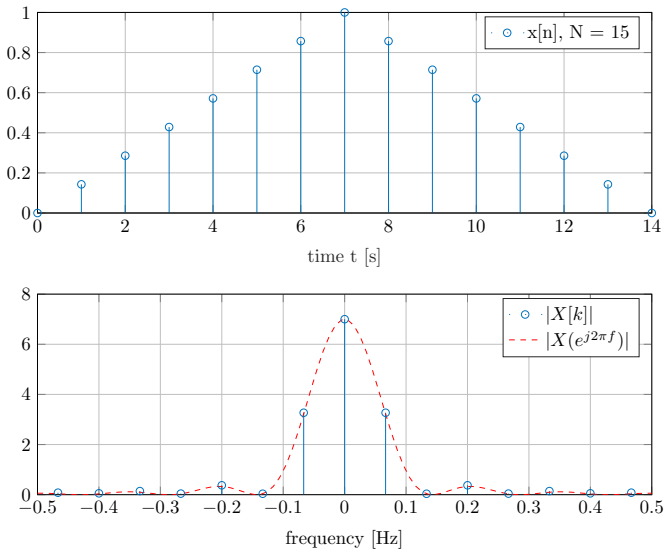

Fig. 6: *DFT* without the effect of the zero padding method.

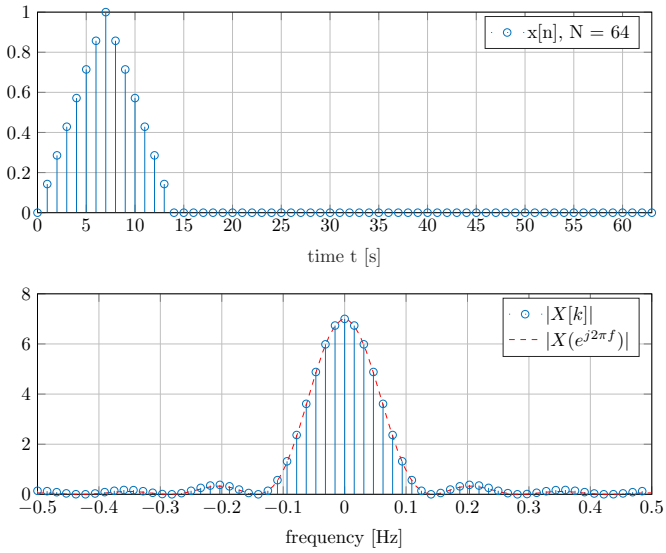

Fig. 7: *DFT* working with N = 64 samples using the zero padding method.

On this exercise a digital signal *x[n]* was defined from the expression:
$x[n] = 2cos(2\pi f_0 n) + cos(2\pi f_1 n)$, where f_0 = 8 Hz and f_1 = 10 Hz.
The sampling was made using a sampling frequency, fs = 32 Hz, and a total number of samples, N = 128.

The first step of the exercise consisted of

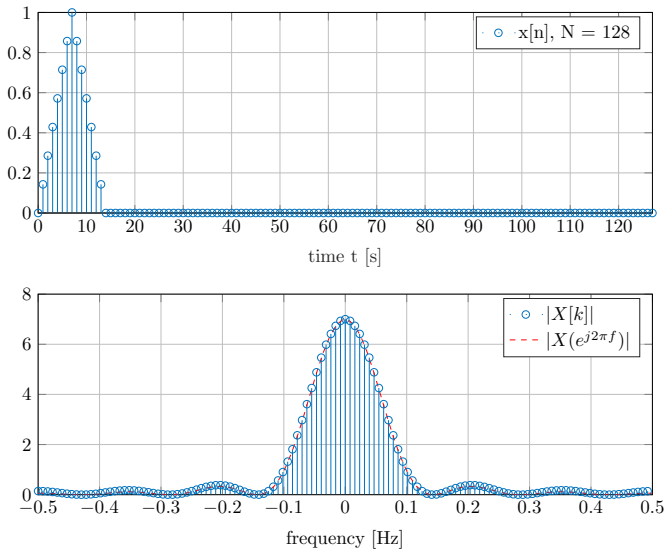

Fig. 8: *DFT* working with N = 128 samples using the zero padding method.

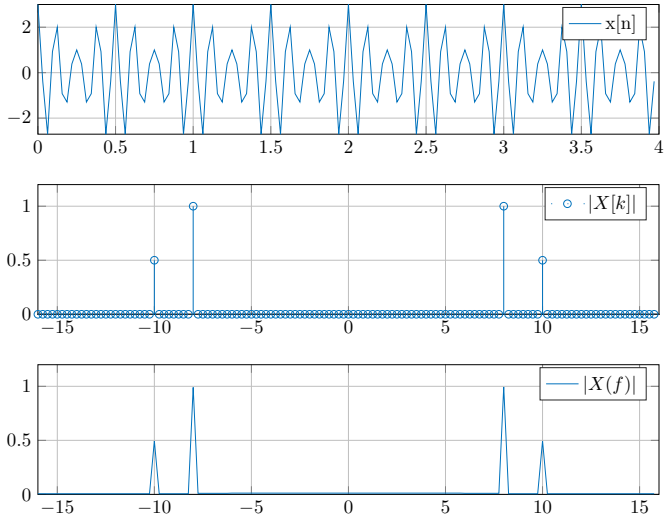

Fig. 9: *DFT* working with N = 128 samples.

calculating the rest of the sampling parameters which were the observation time, $To = N/fc$ and the $\delta f = 1/To$.
Then, to compute the *DFT* of *x[n]*, *X[k]*, and to compare this result with its *CTFT*.

The *DFT* was computed using the function *fft()* while the *CTFT* was computed as the *Fourier Transform* of the signal $r(t) = rect(t/To)x[n]$.
In *Fig.9*. are shown the resulting graphs. As it can be seen, the *DFT* and *CTFT* shows Kronecker and Dirac deltas respectively above $\pm f_0$ and $\pm f_1$. The reason why the f_0 has higher value for its delta derives from the expression x[n], where the first cosine is multiplied by 2.
An important observation is that the value of $To(f1 - f0)$, displayed on the *Command Window*, made reference to an integer value. This means that the sampled signal was representing, in time, an integer number of cycles for both the sin functions. In other words, that it was a multiple of the least-common-multiple of both periods.

In fact, by changing the value of N to 129, can be seen that in *Fig.10* the results showed some differences.
This time the value of *To(f1-f0)* was no more integer, so it means that it was not being sampled correctly the whole period of the signal and that by cutting the signal *x[n]* on that specific point interference was introduced.

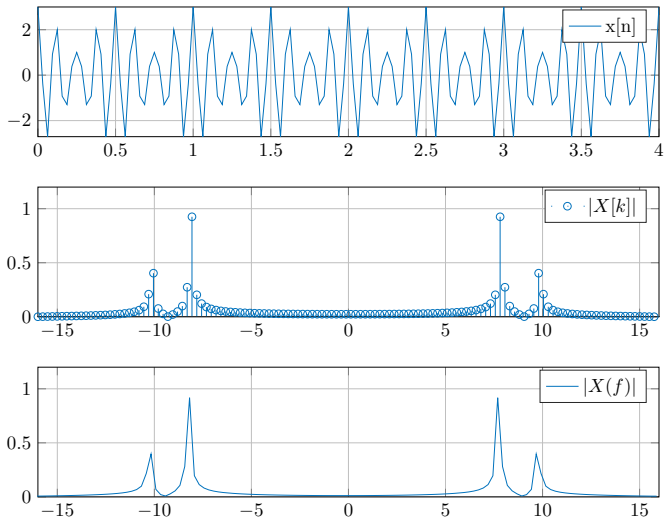

Fig. 10: *DFT* working with N = 129 samples.

Finally the last part of the exercise consisted of zero padding the signal x[n] by adding 128 empty samples.
By doing so, also the values of To and δf changed to be 8 and 1/8 respectively.
It can be noticed that the graphs shown in *Fig.11* are similar to the previous one although this time the value of To(f1-f0) was integer.
The reason why the spectrum is not 'clean' is because by doing the zero padding, a discontinuity was introduced in the signal just after the zero padding starts.

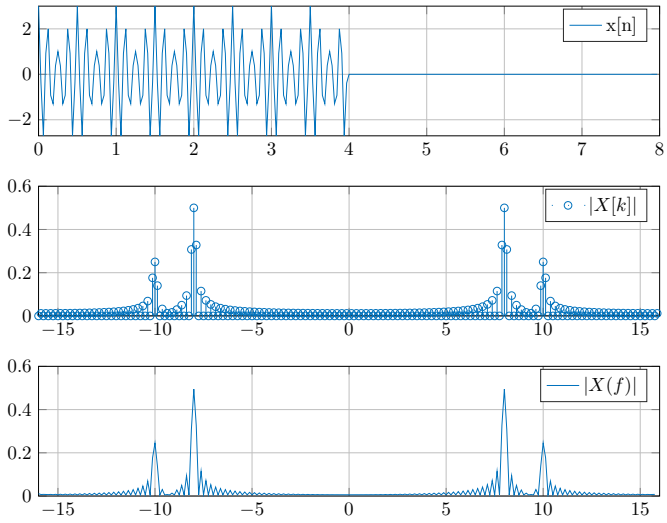

Fig. 11: *DFT* working with N = 128 and zero padding.

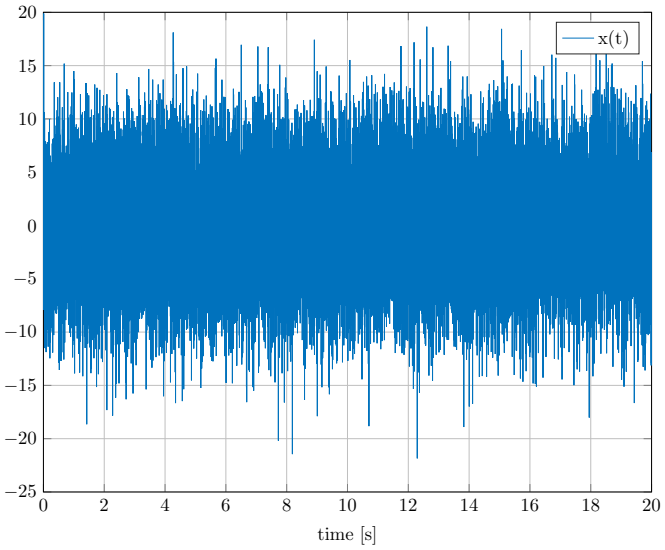

Fig. 12: sinusoidal signal affected by White Gaussian Noise with $\sigma^2 = 25$.

V. Exercise 2.1

ON this exercise the signal:
$x(t) = cos(2\pi f_1 t) + cos(2\pi f_2 t) + W(t)$
having f_1 = 100 Hz and f_2 = 110 Hz, was implemented on *matlab* using a sampling frequency, fs = 1KHz, and observation time, To = 20 sec.
The signal was affected by the presence of a Gaussian noise, W(t), having variance, σ^2 = 25, which was implemented using the *randn()* function. The plot of the signal is shown in *Fig.12*.

The goal of the exercise was to compute the power spectral density of *x(t)*, using the *matlab* function *pwelch()*. Testing it with different parameters and comparing the obtained results.

The idea of *Welch*'s method for estimating the power spectrum is to take the available data, split it up into segments, compute periodogram for each of them and then average the results. By doing so the variance associated with the periodogram can be reduced significantly.

For the first test, it was used a hamming window with length *NFFT* = 128, and no overlap between the segments. *Fig.13*. shows the resulting psd for this test. Can be seen that the psd does not show a high variance as the signal x, instead it is a smoother graph. On the other hand, it does not clearly show the two separate harmonic contributions that should be present avobe $\pm f_1$ and $\pm f_2$.

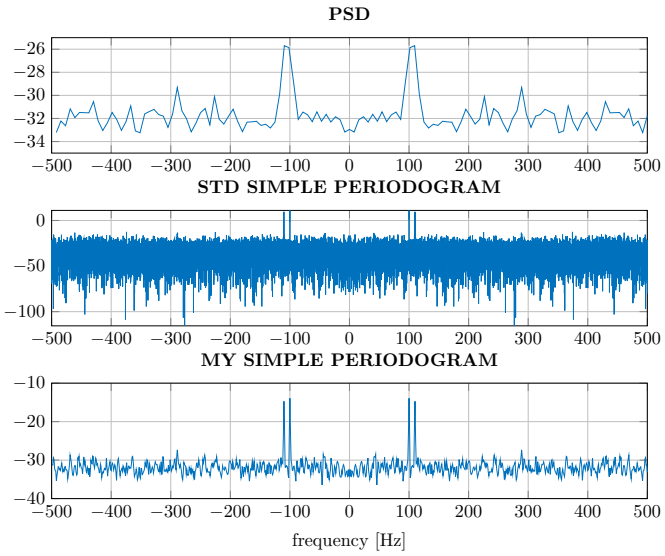

Fig. 13: Power Spectrum Densities using *pwelch()*

This happens because by using more segments with shorter *NFFT*, the effect of applying the hamming window to each of them is to reduce the variance but it also reduces the frequency resolution.
On the contrary, the second graph was obtained with a single segment of length *NFFT* = N and

shows a high variance spectrum.

The next part of the exercise consisted of choosing the parameters for the psd in order to obtain a low variance without loosing a too much of the spectral resolution. The chosen parameters were, a *hann* window, with M = 25 in order to have *NFFT* as the 4% of N. And an overlap of the 60%. The results are shown on the third graph of *Fig.13*.

Finally the last part of the exercise consisted of plotting the *Bartlett* and *Welch* periodograms using in both the cases M = 25, but in the case of *Welch* periodogram use also a 50% overlap and a Hamming window. All the plots are shown in *Fig.14*.

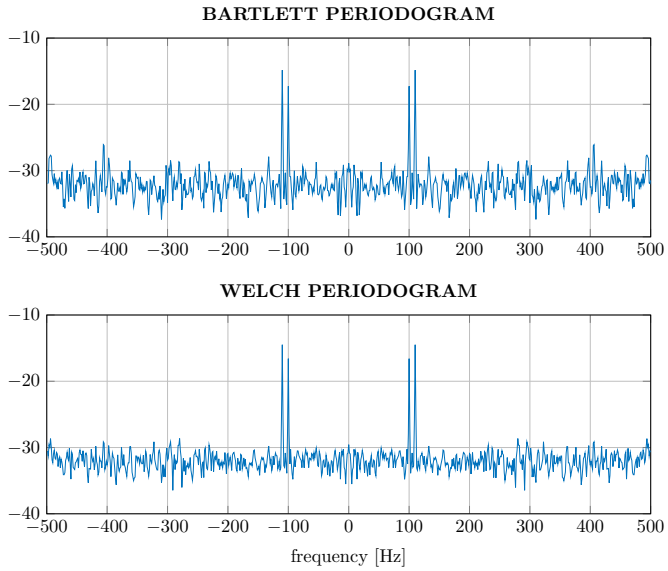

Fig. 14: Comparison between Bartlett's and Welch's periodograms.

VI. Exercise 2.2

FOR the second part of this exercise the signal *x(t)*, described before, was filtered with a low-pass filter obtaining the signal y(t). In *Fig.15*. can clearly be seen the differences between the signal x,y and the sinusoidal function of x without the white noise.

Then the psd was obtained from the three methods listed before. These were the simple periodogram, the Bartlett peridogram with M = 25

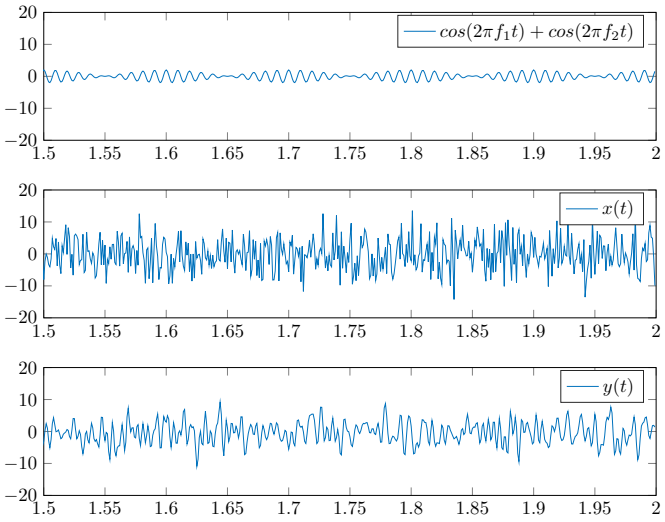

Fig. 15: Comparison between signal x[n] and its filtered version, y[n]

and the Welch periodogram using 50% overlap and different types of windows.

In *Fig.16* can be seen each of the obtained results. As can be noticed the Welch periodogram with the Hann window shows the best ratio between variance and out-of-band attenuation. While the simple periodogram shows a high attenuation but also the highest variation of the set.

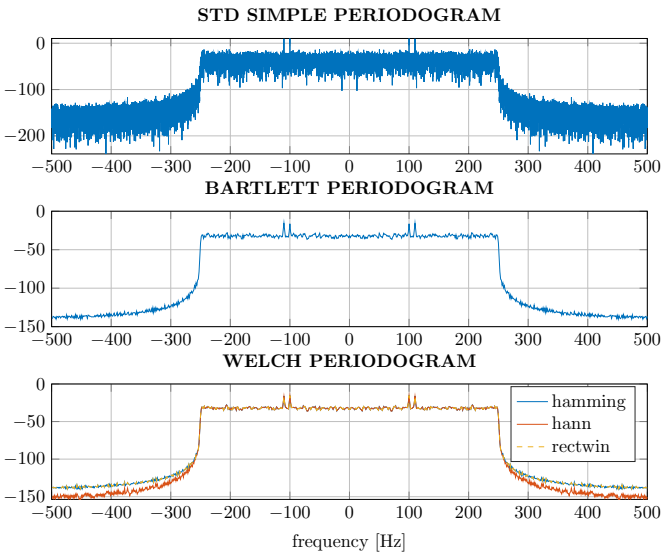

Fig. 16: Comparison between Bartlett's and Welch's periodograms.

VII. Exercise 2.3

FINALLY the last part of the exercise consisted of the implementation of the function *my_Bartlett()* that computes the Bartlett periodogram of a signal x. The function receives as parameters the signal x, its length N, the number of segments M and the sampling frequency fc.

At the beginning, the function checks if N is a multiple of M using the function *mod()*. If this is not the case the signal x is zero padding until it reaches the proper length, using the function *lcm()*.

Then by each iteration on a for loop a sequence of D = N/M samples is computed by the average periodogram. In this process the function *my_DFT()* is called and then is obtained the squared magnitude of the resulting vector, normalizing it by D. At the end of the iterations the function mean() is called.

Lastly the obtained signal is returned with a normalized vector of frequencies going from -0.5 to (0.5-1/D).

A third script was implemented containting the same signals x and y described for the exercise 2.2. The goal of this script was to test the function *my_Bartlett()* and to compare the results with the official *pwelch()* function using a rectangular window with no overlap.
As can be seen in *Fig.17* the obtained psd have the same behaviour as it was expected.

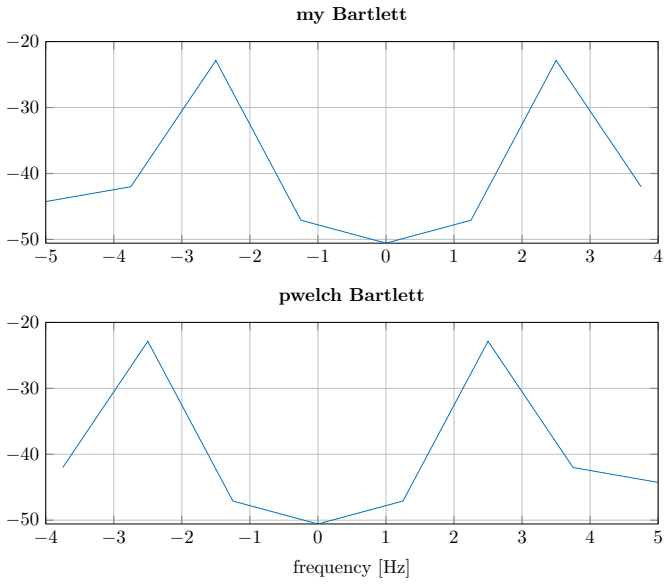

Fig. 17: comparison of y's psd from *my_Bartlett()* and *pwelch()* functions.

Digital Filtering

Juan Manuel Aragon Armas, s253163

Prof. Daniel Gaetano Riviello

***Abstract*—A digital filter in signal processing is commonly defined as a Linear-Time-Invariant system, having a specific transfer function, H(z). It can also be categorized as FIR or IIR, depending on the characteristics of its impulse response.**
When designing filters some properties are studied, such as the band-pass ripple, stop-band ripple, band attentions and transition band.
In this laboratory, the impulse response of some filters were analysed. And on a second part, IIR filters were implemented, using *matlab*, according to some given specifications.

I. Exercise 1.1

FOR the first exercise a set of vectors A and B were given, representing the coefficients of three different IIR filters.

In order to determine which of the filters were stable. The first step was to calculate the roots of each polynomial using the *matlab* built-in function *roots()*. Then those results were plotted using the *zplane()* function.
Knowing that a stable filter has all its poles inside the unitary circle, was easy to determine that the second filter of the set was the stable one.

The next step of the exercise was filtering a Kronecker delta by each one of the filters and then to plot the obtained results using *stem()*.
Also comparing the output signals with the ones showed by *impz()* sending by parameter the set of coefficients of each filter.

In *Fig.1*, *Fig.2* and *Fig.3*. are shown the results of each of the filters as well as their zeros and poles mentioned before.
Can be noticed that only the second filter has all its poles inside the unitary circle, and as it was expected, the output signals tend to zero while going far from the origin.

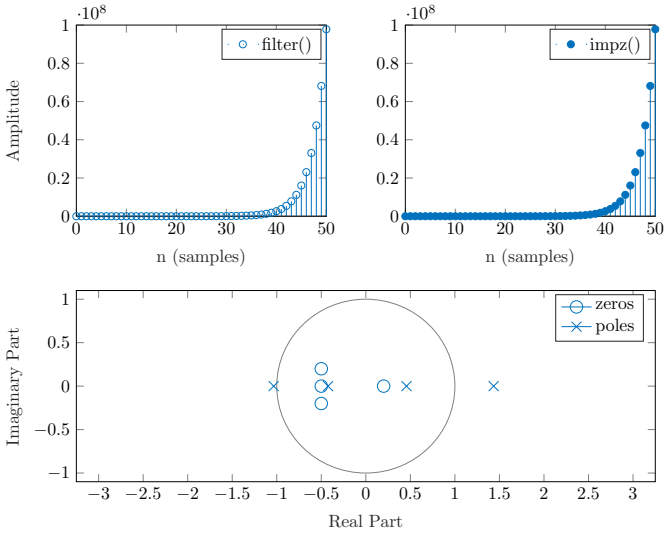

Fig. 1: filter with coeff. B = [1 1.3 0.49 -0.013 -0.029], A = [1 -0.4326 -1.6656 0.1253 0.2877];

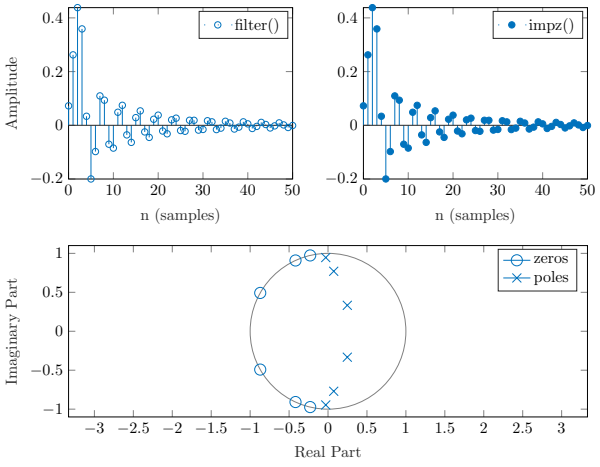

Fig. 2: filter with coeff. B=[0.0725 0.22 0.4085 0.4883 0.4085 0.22 0.0725]; A=[1 -0.5835 1.7021 -0.8477 0.8401 -0.2823 0.0924] ;

Instead, as it was supposed from the roots, the first and third filters are unstable because some of their poles are outside the unitary circle and also because their output signals diverge when a stable

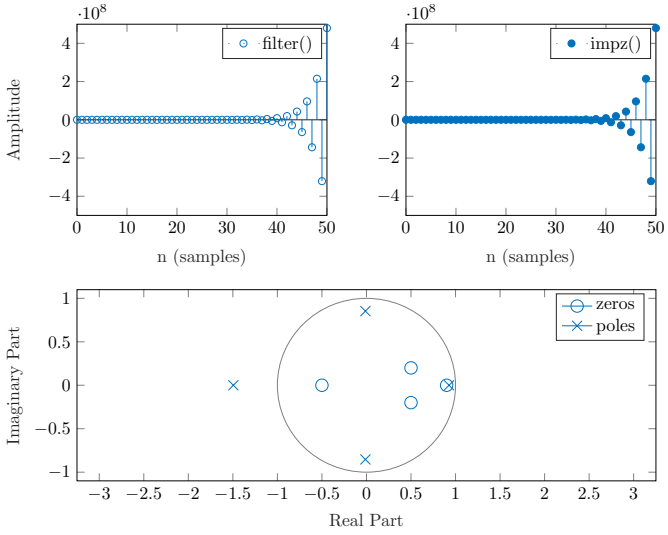

Fig. 3: filter with coeff. B=[1 -1.4 0.24 0.3340 -0.1305]; A=[1 0.5913 -0.6436 0.3803 -1.0091];

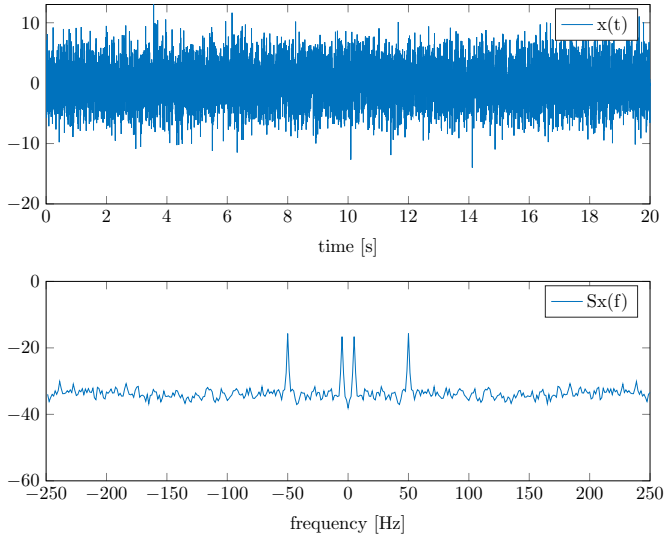

Fig. 4: Power spectral density of x(t)

input signal was given, as the Kronecker delta.

Finally, it can be seen that there is no substantial difference between the plots obtained from the delta and the ones from the *impz()*. This was expected, knowing that the impulse response actually computes the output signal of the filter when an instantaneous impulse is simulated as an input signal.

II. Exercise 1.2

FOR this exercise a signal, $x(t) = cos(2\pi f_1 t) + cos(2\pi f_2 t) + W(t)$, was implemented on *matlab*, setting f_1 = 5 Hz, f_2 = 50 Hz, *To* = 20 s. and f_c = 500 Hz. Being $W(t)$ a zero-mean white Gaussian noise with variance σ^2 = 10 generated with *randn()*.

The first step of the exercise consisted of plotting the power spectral density of *x(t)* obtained from *pwelch()* using 25 segments, a hamming window and 50% overlap.
As can be noticed from the results shown in *Fig.4*, the graph of *Sx* has four peaks above the frequencies $\pm f_0$ and $\pm f_1$.

The next step consisted of filtering the signal *x(t)* using a *butterworth band-pass filter* centered at f_2. The pass-band, *Wp*, of 10 Hz, was defined as the range of frequencies going from 45 to 55.
The stop-band, *Ws*, was defined from the range of frequencies 35 to 65 in order to obtain a transition band of 10 Hz.
Lastly, the band pass ripple and the stop band attenuation had the values 1 dB and 30 dB respectively.

In order to obtain the order of the filter it was used the function *buttord()* and the result was then used to design the filter using *butter()* and *freqz()*.
In *Fig.5* can be seen the shape of the filter in a normalized frequency axis.

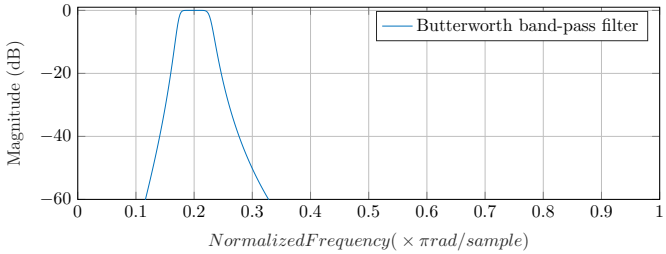

Fig. 5: Butterworth band-pass filter

Then using the obtained filter, it was generated the signal $y_1(t)$ from the signal *x(t)*. In *Fig.6*. can be seen its psd, calculated using the *pwelch()* function with the same parameters mentioned before.
The differences from *y(t)* and *x(t)* are more

noticeable from their psd graph.
It can be seen that because of the filter, it does not show anymore the 5 Hz peak and the rest of the signal is affected by a high attenuation.

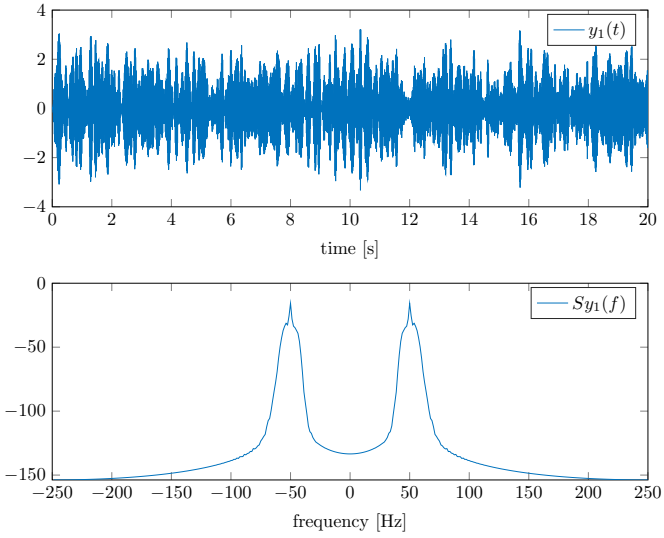

Fig. 6: Power spectral density of $y_1(t)$

The same process was then executed using an *Elliptic* filter instead.
With the functions *ellipord()* and *ellip()* it was also obtained the plot shown in *Fig.7*. And later on, the signal $y_2(t)$ was calculated using *pwelch()*, as can be seen in *Fig.8*.

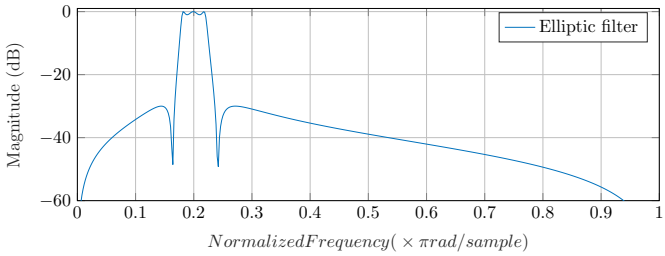

Fig. 7: Elliptic band-pass filter

Comparing the output signals obtained, it can be said that the Butterworth filter does not show a notorious amplitude ripple in either the pass-band and the stop-band. And also that it has a stronger attenuation of the signal near the stop-band.
On the other hand, the elliptic filter is more selective by having a steepest slope from pass-band to stop-band and also needs a lower order to meet

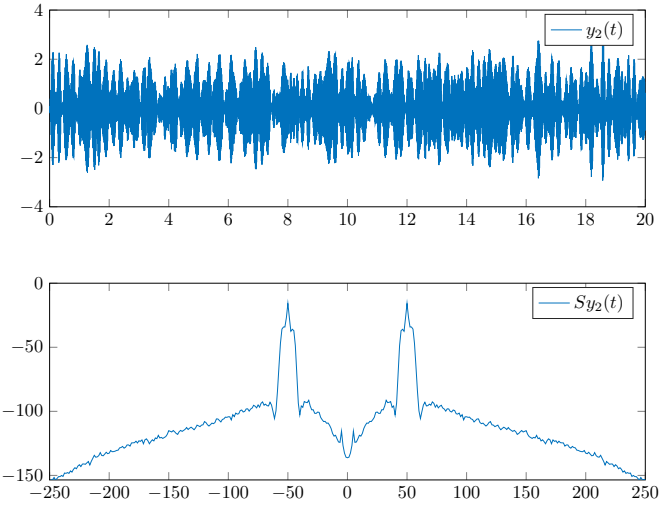

Fig. 8: Power spectral density of $y_2(t)$

the requirements.

Finally, the same process was repeated one more time using the *Chebyshev I* filter.
This time some of the parameters were modified in order to maintain only the 5Hz frequency, such as *Wp* = 7Hz, *Ws* = 12Hz and *Rs* = 60 dB.
The filter diagram can be seen in the *Fig.9* as well as the output signal $y_3(t)$ in *Fig.10*.

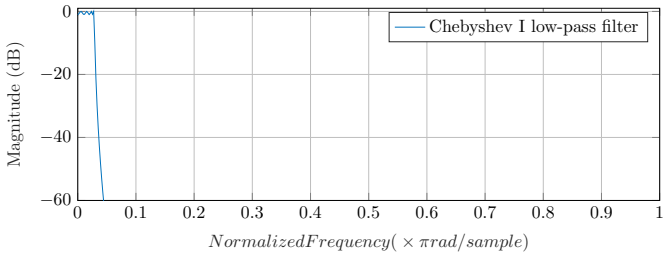

Fig. 9: Chebyshev I low-pass filter

In *Fig.11* is shown how by reducing the variance on the signal x(t), its behaviour is more likely to the signal $y_3(t)$.

III. Exercise 2.1

ON the first part of this exercise, it was studied the mathematical expression that describes a

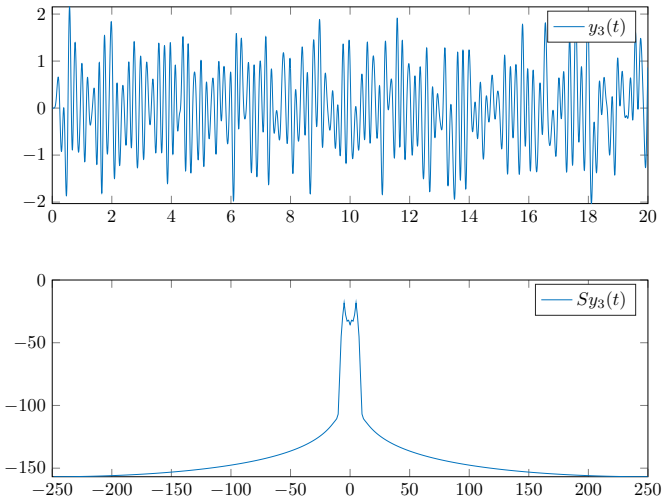

Fig. 10: Power spectral density of $y_3(t)$

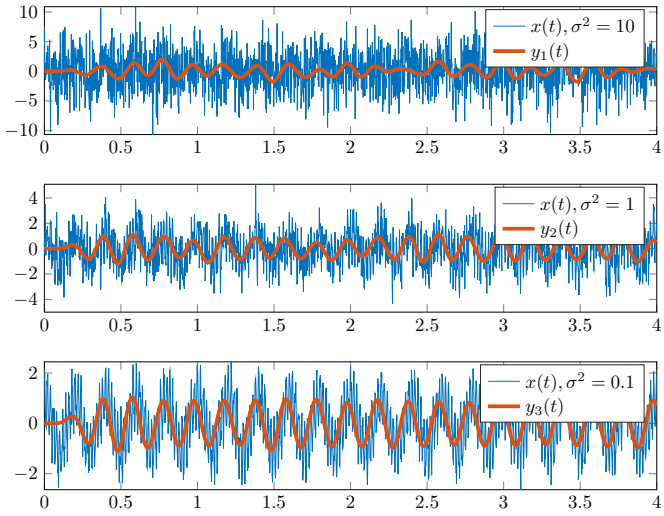

Fig. 11: Comparison between x(t) and $y_3(t)$, changing the STD.

Gibbs phenomenon graphs, as seen in class. The implemented expression was

$$X_T(f) = \frac{Si(\pi T(f + \frac{B}{2})) - Si(\pi T(f - \frac{B}{2}))}{\pi}. \quad (1)$$

Where Si() represents the sine integral function, that on *matlab* was implemented using *sininit()*.

By increasing the parameter T, can be seen from its graph (*Fig.12*), that its behaviour gets every time more similar to a rectangular pulse. However they differ at the vertices, where the implemented expression shows a peak with constant height no matter which values of T are being used.

For the second part of the exercise was studied

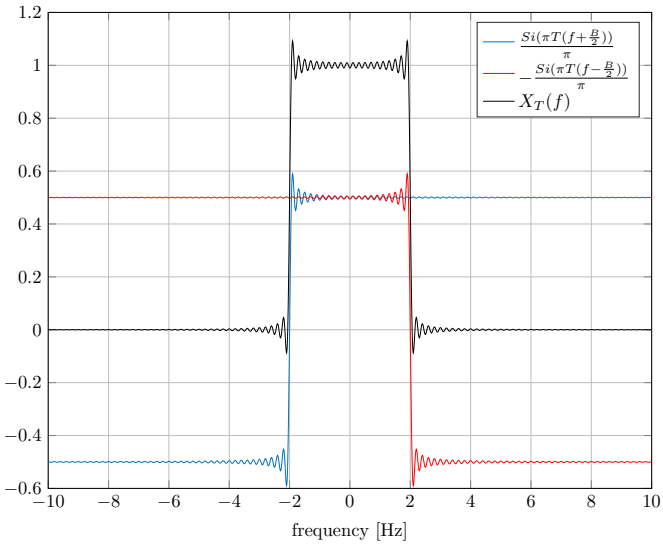

Fig. 12: Mathematical expression of the Gibbs phenomenon

the difference between the mathematical expression seen before and a *DFT* of a *sinc()* function, where the Gibbs phenomenon must be present.

At the beginning, a signal $x(t) = Bsinc(Bt)$ was implemented on *matlab* using the same parameters of the previous exercise, and a sampling frequency, *fc* = 20 Hz.
Then the signal was zero-padded in order to reach a specific amount of 10N samples, being N the initial length of the digital signal x.
Then the rest of the sampling parameters were determined in order to calculate the *DFT* of the signal x.
In *Fig.13* is shown the comparison of the two obtained signals. As can be seen the signals are almost identical, meaning that the precedent math expression describes perfectly the Gibbs phenomenon present on the signal X.

IV. Exercise 2.2

THE current exercise was focused on the design of *FIR* filters by using the *Window Method*.

The goal was to define a low-pass filter having pass-band frequency, $\omega_p = 0.2\pi$, stop-band frequency, $\omega_s = 0.3\pi$, minimum stop-band attenuation, *As* = 40 dB and a maximum pass-band ripple, *Rp* = 0.2

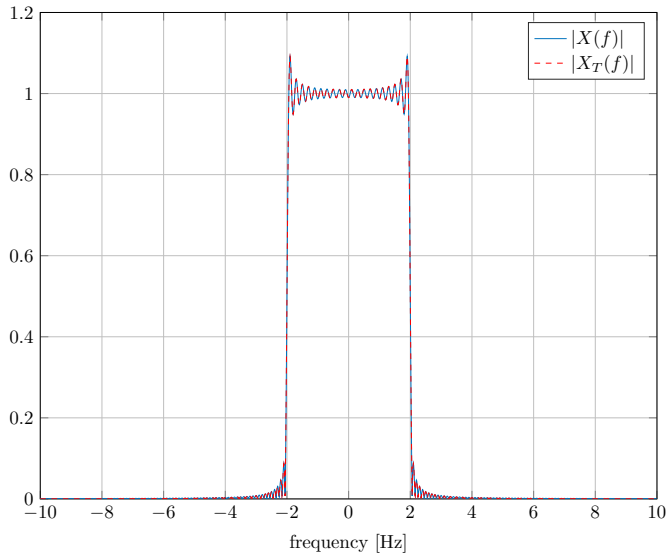

Fig. 13: Comparison between $|X_T(f)|$ and $|X(f)|$

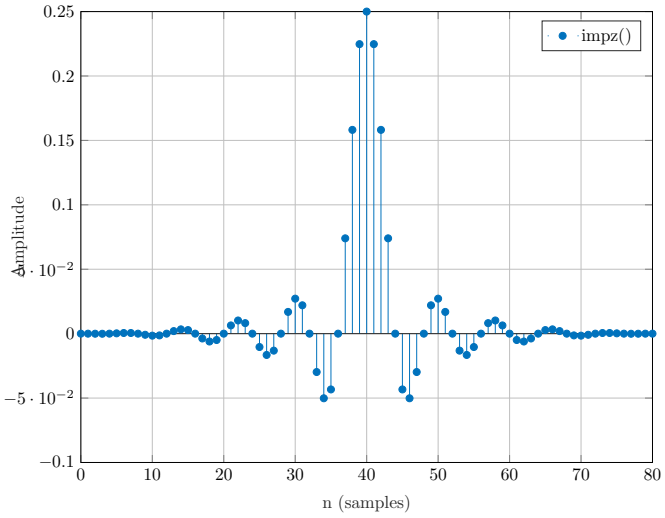

Fig. 14: Impulse response of the FIR filter

dB.

The first step consisted on selecting the window that met with the requirements of *As* and the transition band, $B_T = \omega_s - \omega_p$.
From the As requirement the possible windows to apply were *Hann, Hamming and Blackman*. But as it was necessary to get the lowest number of coefficients, then *Hann* window, *w(n)*, was chosen.

$$w = 0.5 - 0.5cos\left(\frac{2\pi n}{N}\right) \tag{2}$$

At this point the length of the window was defined as $N = \frac{6.2\pi}{B_T}$, as well as the cut-off frequency, *fc*.

$$f_c = \frac{1}{2\pi}\frac{\omega_p + \omega_s}{2} \tag{3}$$

For the next step the vector of coefficients was computed as $h(n) = h_i d(n) w(n)$.

$$h_{id,lowpass}[n] = \frac{\omega_c}{\pi} sinc\left(\frac{\omega_c(n - \frac{M}{2})}{\pi}\right) \tag{4}$$

In *Fig.14* is shown the impulse response of the filter and can be noticed that it has a *sinc()* shape, as it was expected.

On the other hand, in *Fig.15* is shown the amplitude and phase diagrams of the obtained filter. It can clearly be seen that the requirements made on the pass-band frequency fp = $\frac{wp}{2\pi}$ = 0.1 as well as the stop-band frequency fs = $\frac{wp}{2\pi}$ = 0.15 are satisfied.
Also on *matlab*, by zooming on the pass-band region could be noticed that the pass-band ripple *Rp* stayed near the 0.05 dB meaning that indeed all requirements were satisfied.

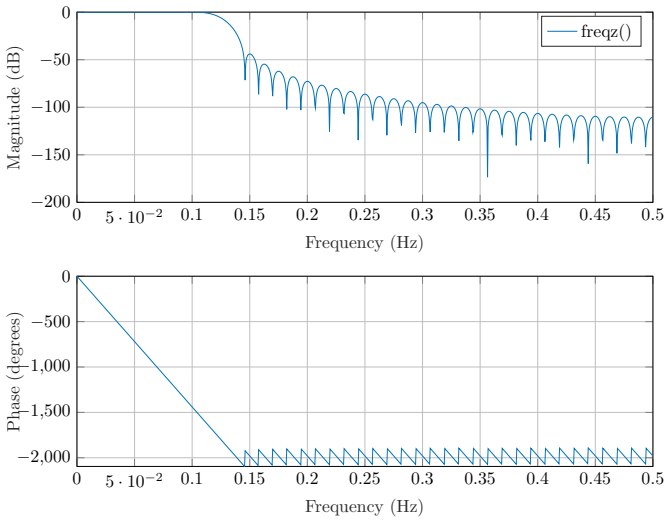

Fig. 15: Amplitude and phase diagrams

V. Exercise 2.3

THE last exercise consisted on implementing the *my_Kaiser_Filter()* function, in order to compute the FIR filter coefficients, *h[n]*, using the Kaiser Window method.

The function received as parameters the values of As, B_T, fs, fc and $type$.
Where the last one, specified the nature of the filter, being low-pass $' - lp'$, high-pass $' - hp'$, band-pass $' - bp'$ or band-stop $' - bs'$. The rest of the parameters used the same notation of the previous exercises.

The first step was to normalize the received fc and B_T by dividing each of the values with the sampling frequency. Then in the case of B_T, it was multiplied with 2π in order to express it as an angular frequency.

The next step was to calculate the parameter beta, that controls the stop band attenuation on the Kaiser method.
It was done by using an if-else statement based on the value of As.

The value of N was calculated using the stop-band attenuation as well as the transition band. In this part the function *ceil()* was used to round the result towards positive infinity.

In order to compute the vector $w[n]$, it was used an array, n, of N values going from 0 to N-1, the parameter beta and the *matlab* function *besseli(0,x)*. Implementing the following expression.

$$w[n] = \frac{I_o\left[\beta\sqrt{1-\left(\frac{2n-N+1}{N-1}\right)^2}\right]}{I_o\left[\beta\right]} \tag{5}$$

And finally according to the type of the filter a different expression for h_d was implemented. Obtaining $h[n] = w[n] * h_d[n]$.

This function was then tested by comparing the frequency response of h[n] with another one obtained by using the function fir1(). The parameters were As = 40dB, B_T = 400Hz, fs = 8 kHz, fc = 800Hz and type = $' - lp'$.

As can be seen in *Fig.16*. both plots show the same behaviour, meaning that the my_Kaiser_filter() function worked correctly. By analysing the graphs can clearly be seen that they correspond to a low-pass filter as expected from the parameter type. Also the cut-off frequency seems to be correct being around the 800Hz.

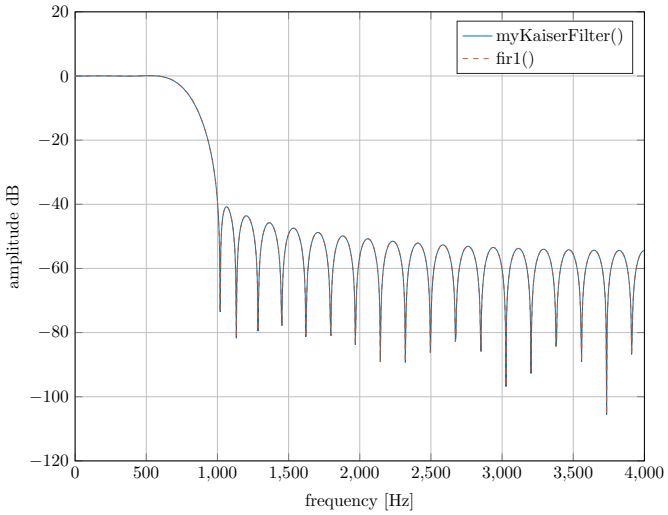

Fig. 16: Amplitude diagram

Processing of Bio-Signals

Juan Manuel Aragon Armas, s253163

Prof. Roberto Garello

***Abstract*—Now a days, technology allow us to obtain sets of data from multiple devices and for different scopes. Some are focused on health, trying to improve the analysis of different parts of the human body. In this laboratory some real-life applications of the signal processing were implemented. In one exercise to analyse the level of oxygen in the blood, and in another, to identify the layers of the cornea and its corresponding thicknesses.**

I. Exercise 1.1

THE first exercise consisted of a digital signal processing application for pulse oximetry. Specifically it was about calculating the blood oxygen saturation and the heartbeat rate, from a stream of data obtained from a pulse oximeter by the emission and reception of two signals of light, red and infrared.

Initially all the pulse oximeter data was contained on a text file, where samples were obtained at a sampling frequency of fs = 100Hz.
The first step consisted of importing the data into *matlab* and to differentiate it into red and infrared samples. Skipping the first 10 sec. of the samples and working only with the next 60 sec.

For the second step, a low-pass filter was implemented using the butter() function with a pass-band frequency, fp = 3Hz, and the same techniques studied on the previous assignments.
In *Fig.1* is shown the amplitude diagram of the implemented filter. The goal was to use it to remove from the input signals all the frequencies higher than the maximum pulse rate, which usually is around 180 bpm.
The obtained filter was then applied to both signals by using the function *filtfilt()* so that the final signal had the same position on time axis.
The results are shown in *Fig.2*. and *Fig.3*.

For the next step it was calculated the heartbeat rate by computing the mean distance between all

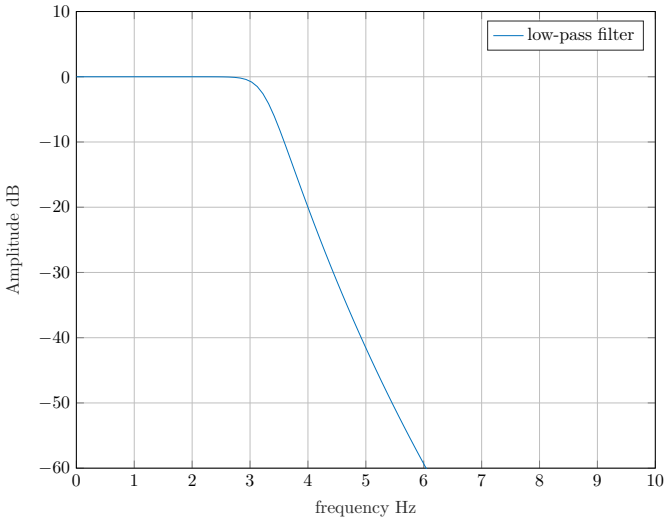

Fig. 1: Butterworth low-pass filter

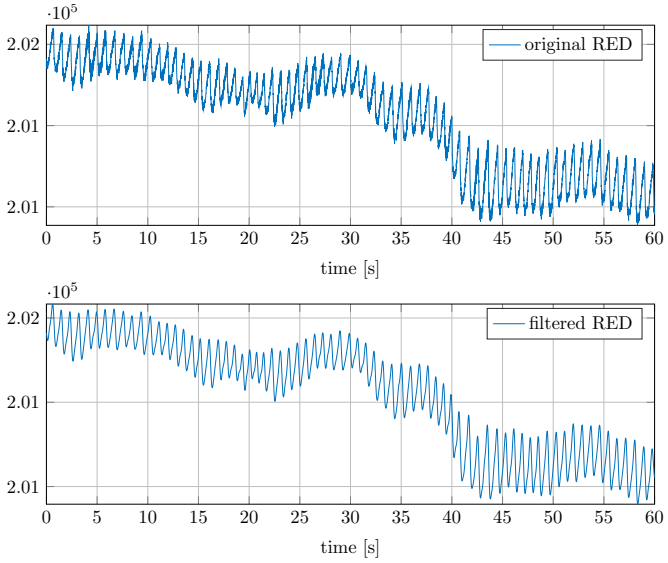

Fig. 2: Filtered Red light signal

the local maximum peaks of the signal. It was done with both signals and the same result was obtained, as can be seen from *Fig.4*. and *Fig.5*.

On the other hand, for the oxygen saturation

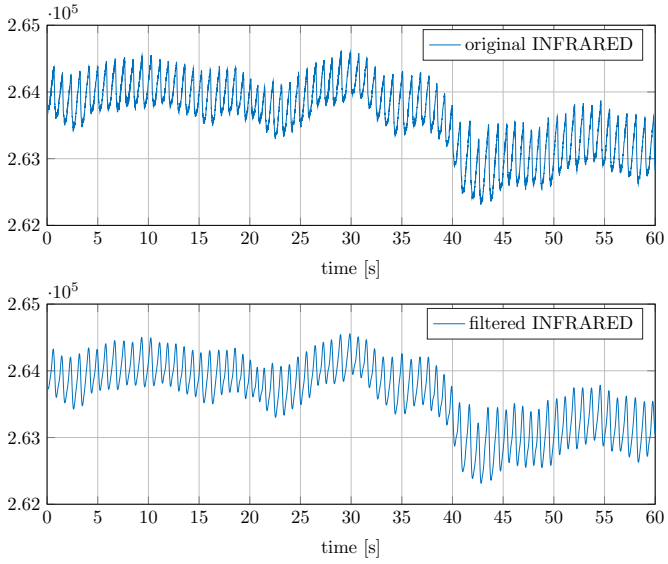

Fig. 3: Filtered Infrared light signal

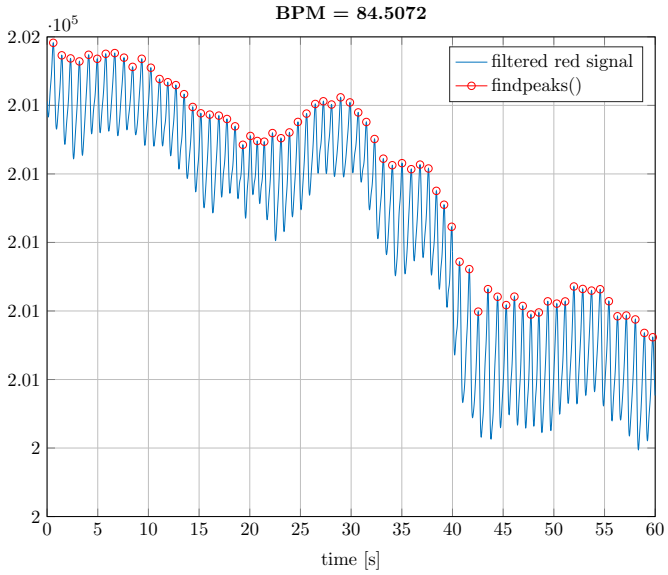

Fig. 4: Heart beat from red light signal

analysis were determined the high and low peaks from each of the signals. Then the interpolating curves were computed by using the function *interp1()*. At this point it was possible to calculate the sampled version of R which mean value is used to calculate the oxygen saturation as follows:

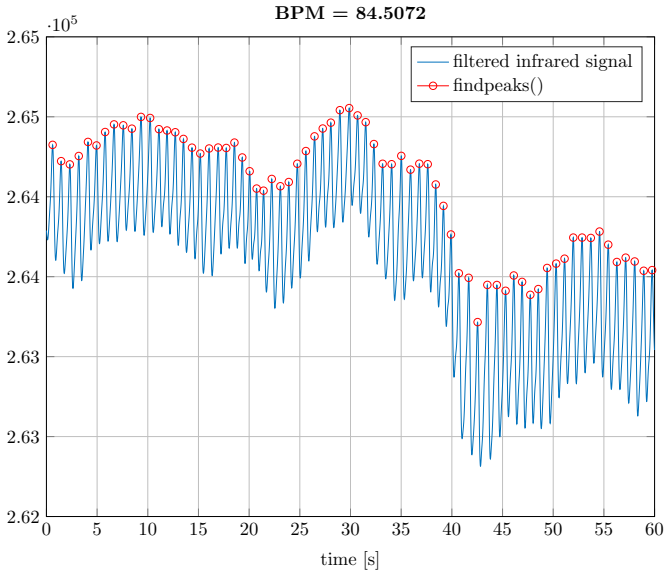

Fig. 5: Heart beat from infrared light signal

$$\bar{R} = \left\langle \frac{\frac{I_{AC}(RED)}{I_{DC}(RED)}}{\frac{I_{AC}(IR)}{I_{DC}(IR)}} \right\rangle \tag{1}$$

$$SaO_2 = 110 - \bar{R} \tag{2}$$

In *Fig.6.* and *Fig.7.* are shown the interpolation curves for both the signals. From which the obtained value of SaO_2 was about 98.16%.

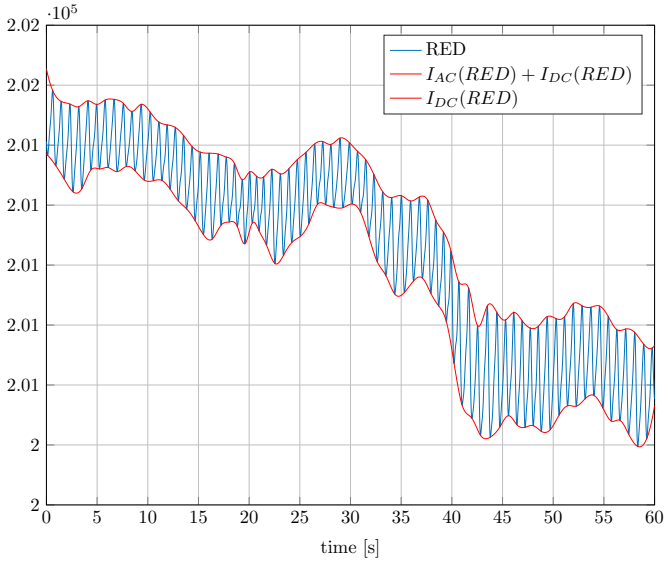

Fig. 6: Interpolation curves for the red light signal

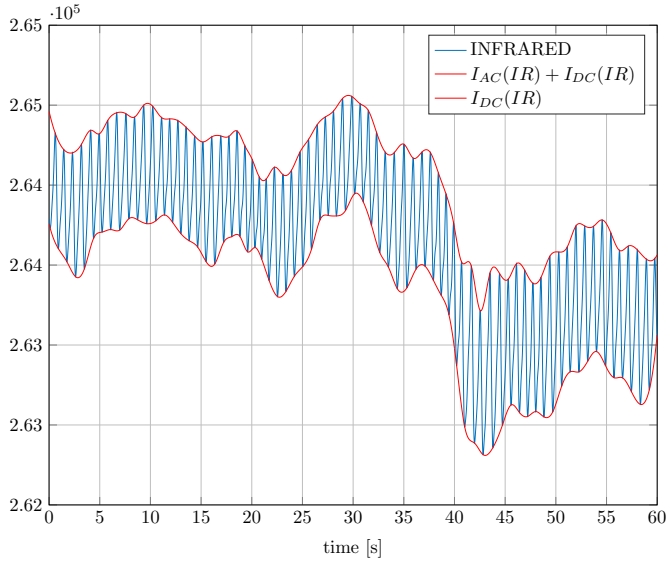

Fig. 7: Interpolation curves for the infrared light signal

II. Exercise 1.2

THE second exercise was about biomedical image processing on *OCT* tomograms.
It consisted of analysing an Optical Coherence Tomography using *matlab*, in order to identify the thickness of all the principal layers of the cornea.

On the first part, the *OCT* image was uploaded and stored into a 640x1016 matrix, having on each cell the corresponding pixel value of the image.
All the values were integers ranging from 0 (black) to 255 (white).

The implemented notation considers each point as p = (x,y), where x refers to the column-index and y to the row-index. Meaning that the origin p_o = (1,1) is located at the top-left corner.

In order to find the cornea axis, which is the center-uppermost point located on the external layer, the image was scanned sequentially from p_o, searching for the first sequence of white pixels, from which it was calculated the middle point. The obtained result was $p_m = (x_m, y_m)$ = (494, 173).

The second task consisted of estimating the radius of cornea.
It was done by choosing a second point, p_a, in the same external layer, having $x_a = \frac{x_m}{2}$.

Then calculating the center of the cornea using pa and pm, supposing the *OCT* as a fraction of a perfect circle.
So $p_c = (x_c, y_c)$ where $x_c = x_m$ and y_c was computed using the formula:

$$y_c = \frac{(x_a - x_c)^2 - y_m^2 + y_a^2}{2(y_a - y_m)}. \tag{3}$$

And finally the value obtained was $R = y_c - y_m$ = 999 pixels = 4.125 mm (supposing 1 pixel = $4.13\mu m$).

For the next task the mean intensity values on the central region were computed.
Considering a lag of 25 pixels and a given x1. It was generated a second matrix with dimensions 640x51 containing the central region of the *OCT*. Then it was computed the mean for every row vector which results are plotted in *Fig.8*.

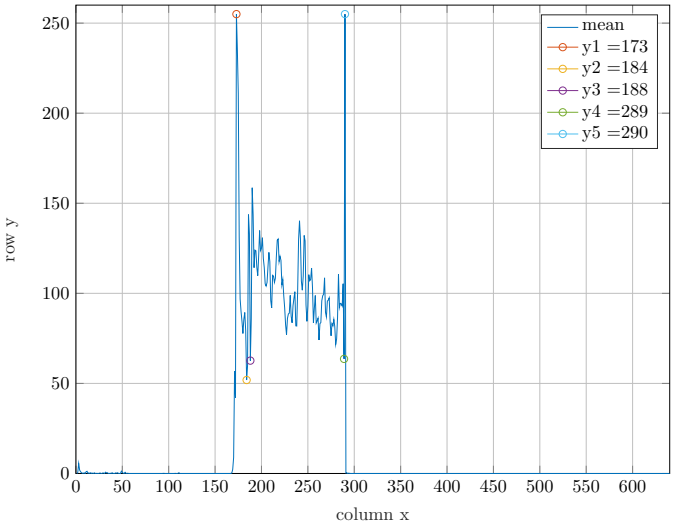

Fig. 8: Mean intensity values on the central region

Can be seen that the *Epithelium* region begins at the highest peak on the left, in the 173rd row.
The *Endothelium* on the other hand ends at the highest peak on the right, at the 290th row.
In between, the lowest peak at y = 184 delimits the *Epithelium* and the *Bowman* regions while the peak next to it, y = 185, divides the *Bowman* and *Stroma* regions. Finally the minimum peak just before y5, marks the end of the *Stroma* and the beginning of

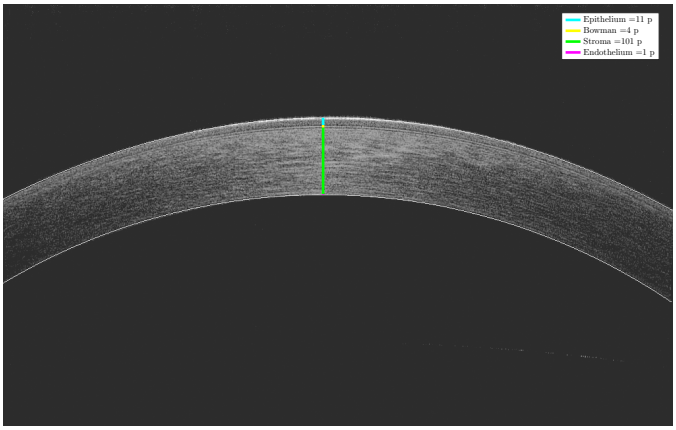

Fig. 9: Position and thickness of the principal layers of the cornea

the *Endothelium*.

In *Fig.9* can also be seen the *OCT* image with all the different regions plotted, describing for each one its length in pixels.