Politecnico di Torino

Applied Signal Processing Laboratory

Assignment 1: Fundamentals of Spectral Analysis

Francesco Laterza, 271087 Giorgio Agnello, 272742

March 2023

Contents

1 Introduction to Signals and Sampling **4**
- 1.1 Exercise 1 . . . 4
- 1.2 Exercise 2 . . . 5
  - 1.2.1 Energy . . . 5
  - 1.2.2 Discrete Fourier Transform . . . 6
  - 1.2.3 Energy contained in lobes . . . 7
- 1.3 Exercise 3 . . . 7
  - 1.3.1 Oversampling with Sampling Frequency 1000Hz . . . 7
  - 1.3.2 Undersampling with Sampling Frequency 160Hz . . . 9
  - 1.3.3 Undersampling with Sampling Frequency 120Hz . . . 9

2 Spectral Analysis of discrete-time signals: DTFT and DFT **10**
- 2.1 Exercise 4 . . . 10
- 2.2 Exercise 5 . . . 13
- 2.3 Exercise 6 . . . 14
  - 2.3.1 Linear convolution . . . 14
  - 2.3.2 Circular convolution . . . 14
- 2.4 Exercise 7 . . . 15
  - 2.4.1 DFT . . . 15
  - 2.4.2 DFT zero-padding . . . 17

3 FFT and Spectral Analysis of analog signals through DFT **17**
- 3.1 Exercise 8 . . . 17
  - 3.1.1 Signal Energy . . . 18
  - 3.1.2 Signal Fourier Transform . . . 18
  - 3.1.3 Bandwidth containing 99.9% of Total Energy . . . 19
  - 3.1.4 Comparison with Discrete Signal . . . 19
- 3.2 Exercise 9 . . . 20
  - 3.2.1 DFT and CTFT comparison . . . 21
  - 3.2.2 Change number of samples . . . 21
  - 3.2.3 Zero-padding . . . 21
- 3.3 Exercise 10 . . . 23
  - 3.3.1 DFT . . . 23
  - 3.3.2 Zero-padding . . . 23
  - 3.3.3 Convolution by Sinc . . . 25

4 Discrete-time correlation functions and random processes **25**
- 4.1 Exercise 11 . . . 25
- 4.2 Exercise 12 . . . 27
- 4.3 Exercise 13 . . . 32

5 Fundamentals of spectral estimation **32**
- 5.1 Exercise 14 . . . 32
  - 5.1.1 PSD estimation . . . 33
  - 5.1.2 Optimized periodogram . . . 33
  - 5.1.3 Periodogram comparison . . . 33
  - 5.1.4 Auto-correlation estimation . . . 34
  - 5.1.5 Filtering the signal . . . 36

5.2 Exercise 15 36
5.2.1 Implementation 36
5.2.2 Testing 37

1 Introduction to Signals and Sampling

1.1 Exercise 1

The object of this exercise is to generate 100 sinusoidal truncated signals

$$s_i(t) = \text{rect}_T(t - T/2)\sin(2\pi f_i t + \phi_i) \qquad i = 1, 2, \ldots, 100$$

For this simulation we consider $T = 1$, $f_i = i$ and ϕ_i a random number uniformly distributed between 0 and 2π. The sampling frequency f_s is chosen to be 20 times the highest frequency of the sine function f_{100}.

The signals are then generated in a for loop and stored in a matrix s of dimensions $100 \times f_s$.

To increase the efficiency the matrix is initialized to all zeros to avoid dynamic memory allocation at each iteration of the for loop.

There is no need to consider the rect function, it is enough to consider only the points in which is non zero, thus from 0 to T.

We know compute 3 sums of sine functions

$$s_{a,b,c}(t) = \sum_{i=1}^{6,24,96} s_i(t)$$

It is enough to apply the Matlab *sum* function to a sub-matrix of s taking respectively rows from 1 to 6,24,96 and all columns.

We can now plot the results

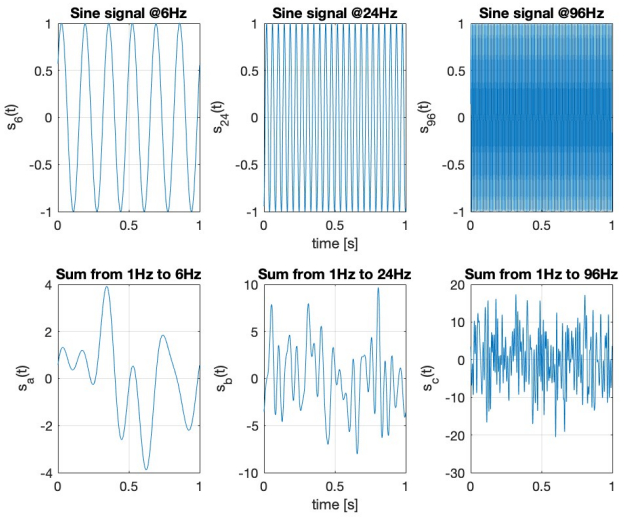

Figure 1: Resulting plots

In Figure 1 the sine function at 6, 24 and 96 Hz are plotted. Also s_a, s_b and s_c are present, which are respectively the sum of the first 6, 24, 96 sine wave functions.

It can be noticed as the more contributions we add, the faster the resulting signal varies. Also because of the presence of a random phase shift for each contribution, the more elements are added the more the signal will be random.

1.2 Exercise 2

For this exercise we have to consider a rectangular pulse

$$x(t) = A\,\mathrm{rect}_T(t - T/2)$$

the time shift indicate that the rectangular function start at $t = 0$.

Parameters the amplitude A and the duration of the pulse T have to be chosen by the user. Required conditions are $A \in [0.1, 10]$ and T integer $\in [1, 5]$.

We then consider a the simulation duration as 10 times the pulse duration. Also we take as sampling frequency f_s to be $1000/T$. Consequently a sampling time t_s of $T/1000$ meaning we will have 1000 samples in the pulse duration.

The rectangular pulse is computed in Matlab with the function *rectangularPulse*. For this simulation A and T are both set to 1

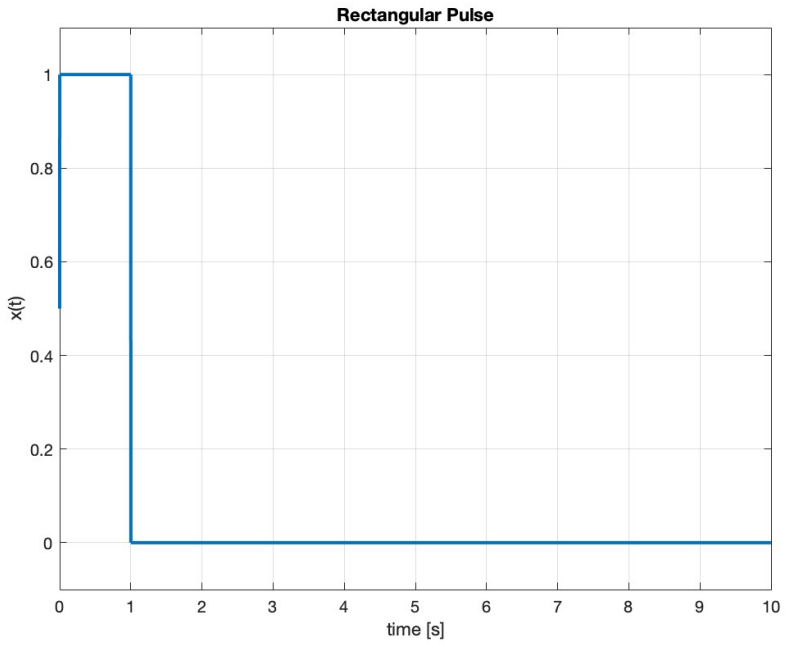

Figure 2: Rectangular Pulse

1.2.1 Energy

The energy of a signal is computed as

$$\mathcal{E}\{x(t)\} = \int_{-\infty}^{+\infty} |x(t)|^2\, dt \tag{1}$$

In Matlab we can compute the integral with the function *trapz*, which compute the area of the trapezoid formed between two consecutive points of the vector.

The **numerically computed energy is 0.99938**. Notice that from theory we know that the energy of the rectangular function is $E_{\text{rect}} = A^2T$, which in our case would result **analytically to 1**. The small difference is due to the fact that the numerical integration applies some approximation. Though the **relative error is only 0.0625%** which is small enough.

1.2.2 Discrete Fourier Transform

The frequency axis in the simulation is a vector with values from $-\frac{f_s}{2}$ to $\frac{f_s}{2} - \frac{f_s}{N}$ with a difference between two consecutive elements of $\frac{f_s}{N}$, where N is the number of samples. The DFT can be computed in Matlab with the command *fft*.

Notice that we need to re-center on the frequency axis with the function *fftshift*.

We can now compute the PSD of the signal as

$$PSD(x) = |X(f)|^2 \tag{2}$$

we apply a normalization such that the maximum value of the PSD is A^2T^2. It is enough to multiply each element of the vector by $A^2T^2/\text{max-value}$.

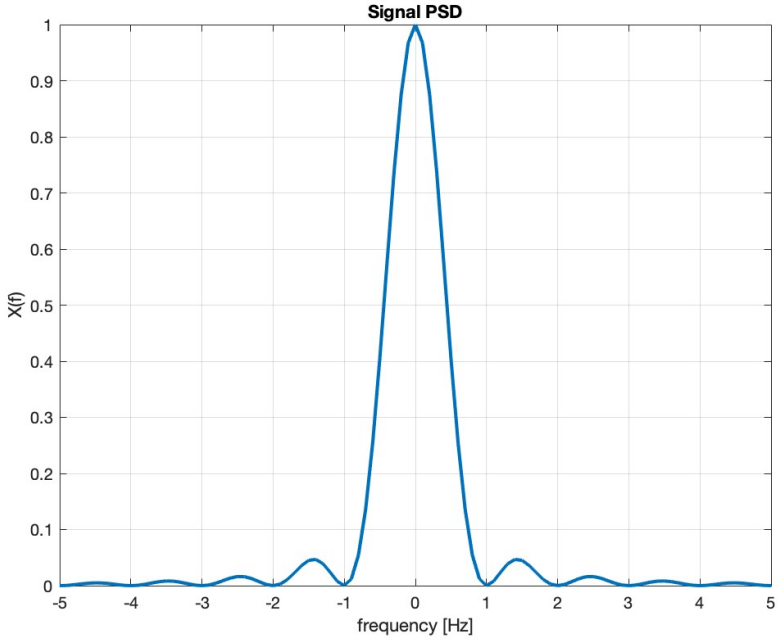

Figure 3: Rectangular pulse PSD

To compute the energy of the signal in the frequency domain we can apply the *trapz* function to the PSD in Figure 3. The **result is 0.9995** and the **relative error with respect to the analytical result is 0.05%**.

1.2.3 Energy contained in lobes

In order to compute the energy stored in each lobe it is necessary to find the indexes of the lobes in the frequency vector. We can do it by finding the index of the zero frequency as $f_{0,idx} = N/2 + 1$ with N the number of samples. We can then compute how many frequency elements there are in between 2 lobes as $step = \frac{1}{T}\frac{N}{f_s}$, where T is the pulse duration and f_s the sampling frequency.

The energy stored in the nth lobe is then

$$\mathcal{E}_n = \mathcal{E}_{n-1} + \int_{f_{o,idx}+((n-1)\cdot step)}^{f_{0,idx}+n\cdot step} PSD(x)\, df$$

it is then possible to compute the percentage of energy contained in a lobe as $\mathcal{E}_{\text{lobe}}/\mathcal{E}_{\text{total}}$

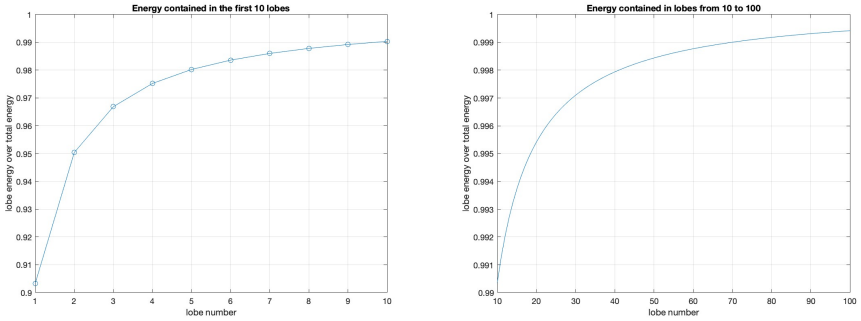

Figure 4: Percentage of energy in the first 100 lobes

In Figure 4 it is noticeable as more than the 90% of the energy is contained in the first lobe as expected. In the first 10 lobes the 99% of energy is stored to arrive at the 100^{th} lobe that contains more than the 99.9% of energy. It is clear that the of the percentage of energy with respect to the lobes number starts very high at the first node and then has a kind of logarithmic increase.

1.3 Exercise 3

We consider a signal

$$x(t) = A\,\text{rect}_T(t - T/2)[\sin(2\pi f_1 t + \phi_1) + \sin(2\pi f_2 t + \phi_2) + \sin(2\pi f_3 t + \phi_3)]$$

where f_1, f_2 and f_3 are respectively 10, 20 and 100 and ϕ_1, ϕ_2 and ϕ_3 are randomly generated numbers from 0 to 2π.

To represent this function in Matlab there is no need to multiply by the rectangular function. It is enough to consider the sum of sinusoidal functions limited between 0 and T, multiplying by the constant A.

1.3.1 Oversampling with Sampling Frequency 1000Hz

The Nyquist Theorem states that in order to safely sample a signal, the sampling frequency f_s should be greater or equal to the double of the signal bandwidth.

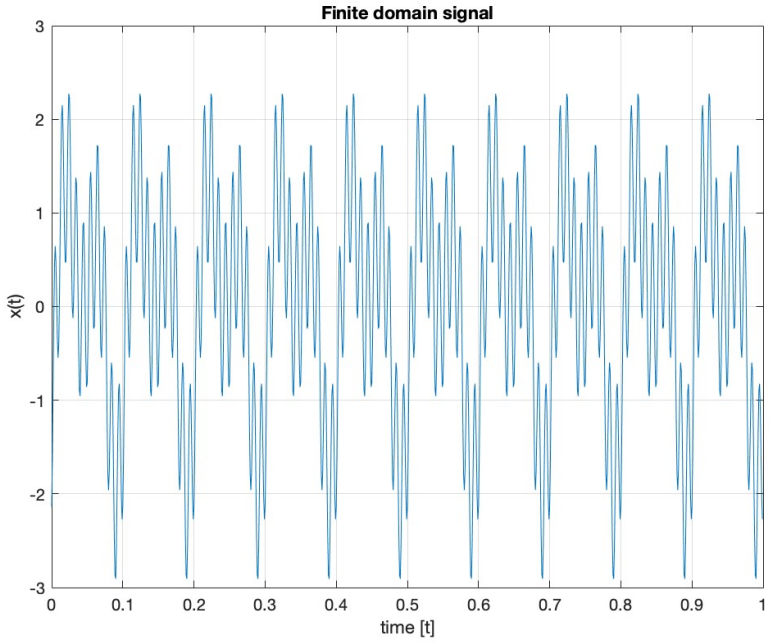

Figure 5: Signal in time domain

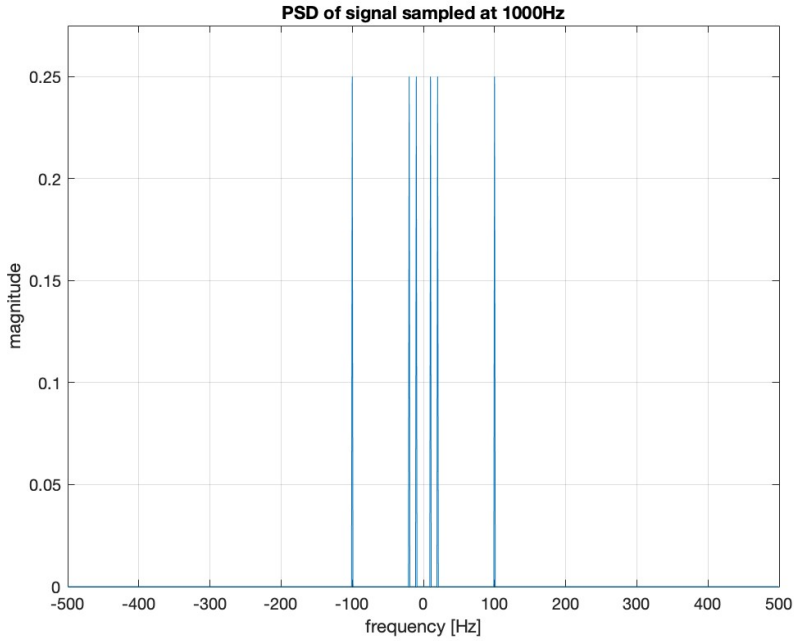

Figure 6: PSD of oversampled signal @1000Hz

In our case the bandwidth is 100Hz. Thus $f_s = 200$Hz would be sufficient. We oversample the signal considering $f_s = 1000$Hz.

In Figure 6 we can see as expected, on the positive frequencies, three deltas at f equal to 10Hz, 20Hz and 100Hz. The spectrum is fully represented without overlapping since the signal is oversampled.

1.3.2 Undersampling with Sampling Frequency 160Hz

Now we consider f_s to be 160Hz, below the safety value of 200Hz.

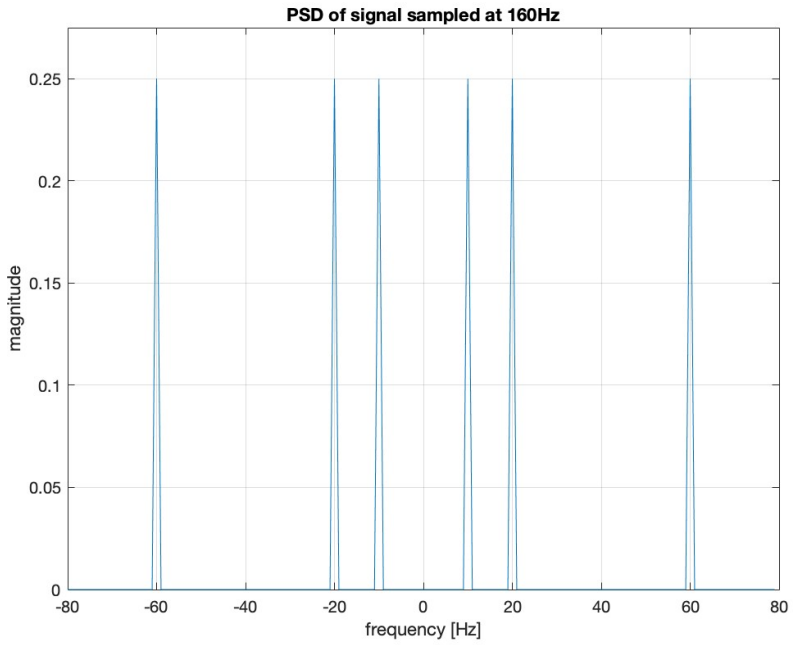

Figure 7: PSD of undersampled signal @160Hz

In Figure 7 on the positive frequency, the deltas at 10Hz and 20Hz are correctly present, though the 100Hz delta cannot be represented since the spectrum ranges up to 80Hz. A 60Hz delta shows up, which did not belong to our signal. This component is due to the undersapling. The sampling frequency is too small, causing an overlapping of the frequency band in the frequency domain.

The 60Hz component comes from the overlapping of the -100Hz component, in fact $f_s - 100Hz \Rightarrow 160Hz - 100Hz = 60Hz$.

1.3.3 Undersampling with Sampling Frequency 120Hz

This time we undersample even more with a sampling frequency of 120Hz.

In Figure 8 we can observe three different simulation undersampling with sampling frequency at 120Hz. This time we see that there are only 2 deltas. One at 10Hz and one at 20Hz. What is interesting is that the deltas have not

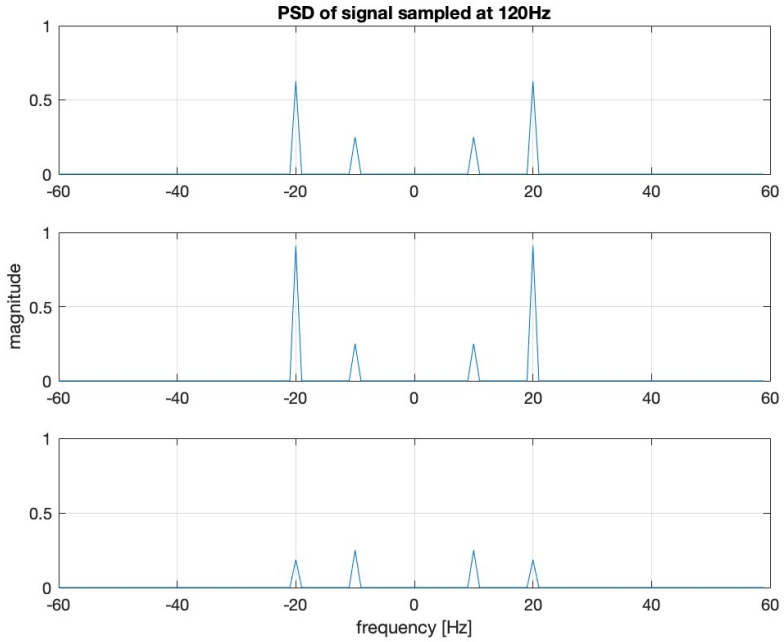

Figure 8: PSD of undersampled signal @120Hz

the same magnitude. Moreover also they are different from one simulation to another.

The reason is that at $f = 20$Hz there are two contributions present. One from the signal itself and the other one from the overlapping of the 100Hz component. Since the frequency of 20Hz and 100Hz have random phase, their sum gives at each simulation a different result, changing the output of the frequency domain.

2 Spectral Analysis of discrete-time signals: DTFT and DFT

2.1 Exercise 4

The Sinc function has DTFT

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

in order to plot it we define the frequency axis using the Matlab command *linspace* to create a vector from -1 to 1 of 1001 points.

We plot the Sinc DTFT for a value of L ranging from 3 to 10, integer number

In Figure 9 notice that increasing the value of L the spectrum components get higher around the component corresponding to a multiple of the Sinc frequency (1Hz). Eventually if we continue to increase the value, we will generate a train of deltas.

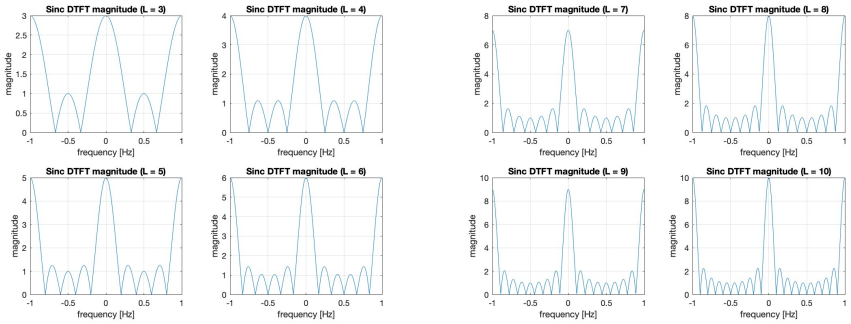

Figure 9: Sinc DTFT with L from 3 to 10

Now we compute the Periodic Sinc function

$$Z(e^{j2\pi f}) = \frac{\sin(\pi f L)}{\sin(\pi f)}$$

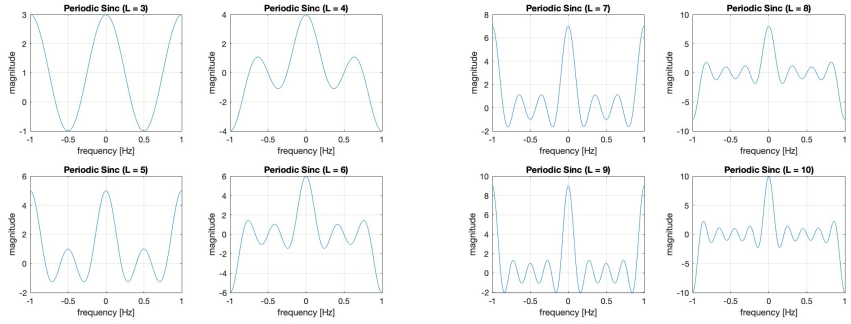

Figure 10: Periodic Sinc with L from 3 to 10

In Figure 10 the Periodic Sinc with L ranging from 3 to 10 is plotted. The period of this function changes based on if L is even or odd. When L is even the period of Z is 2; when L is odd the period of Z is 1.

To compare the Periodic Sinc with a regular Sinc lets define

$$W(e^{j2\pi f}) = \frac{\sin(\pi f L)}{\pi f} = L\,\mathrm{Sinc}(fL)$$

The two functions Z and W are compared for $L = 13$

To better represent the differences lets analyse the Mean Square Error between the two signals. The MSE is defined as follows

$$E(f) = |Z(e^{j2\pi f}) - W(e^{j2\pi f})|^2$$

In Figure 12 it is clear that as we get further from the origin, the error between the two signal increases. This can also be seen in Figure 11, at the origin the functions coincide but as we move towards higher frequencies the start diverging. There are **4 lobes** in the interval from 0 up to $E(f) < 0.16^2$.

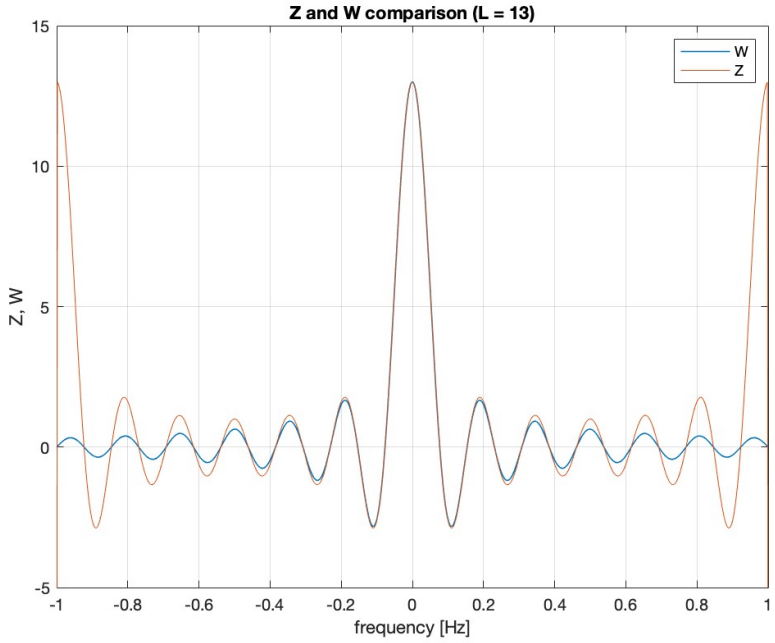

Figure 11: Comparison between Sinc and Periodic Sinc

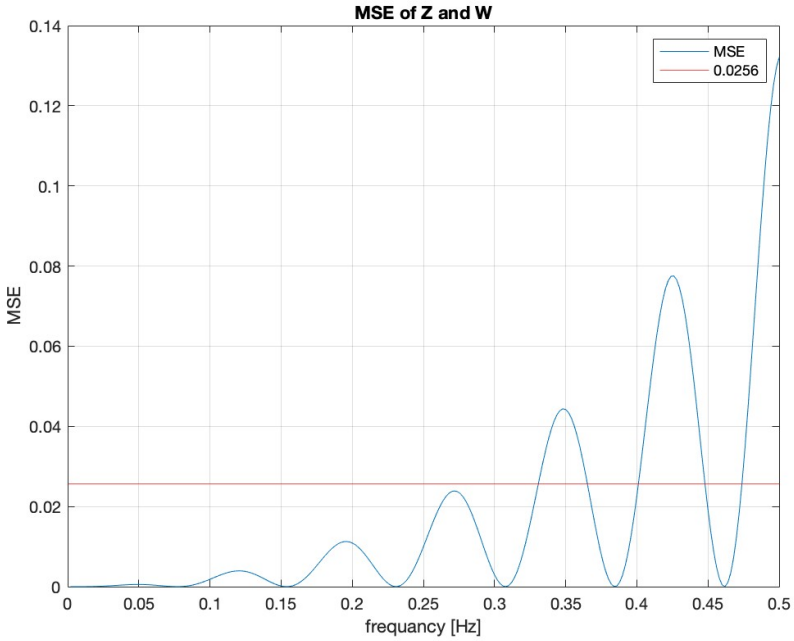

Figure 12: MSE of Z and W in the positive frequency axis

2.2 Exercise 5

This exercise is focused on the implementation of a custom function which computes the DFT of a sequence of N samples.

The mathematical equation of the Discrete Fourier Transform is

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi n \frac{k}{N}} \tag{3}$$

In Matlab we created a function which takes as input parameters the sequence x and the number of samples N and will return the DFT sequence X.

Before computing the DFT we allocate the memory for the resulting vector X defining it as a vector of N zero elements. We also define a vector of integer numbers $n = [0, \ldots, \text{length of } x - 1]$

We can now compute the sum for a fixed value of k. First we compute the product element wise between the vector x and $\exp(-j2\pi nk/N)$ and then the sum of all the components of the resulting vector.

At this point it is enough to repeat this process for all values of k ranging from 0 to $N - 1$.

To move the negative components at the beginning of the vector we use the Matlab function *circshift* with a value of $N/2$. To ensure the correct functioning also in the case N is a odd number it is necessary to consider the *floor* of $N/2$.

We test our custom function with the Matlab *fft* built-in function with a random sequence.

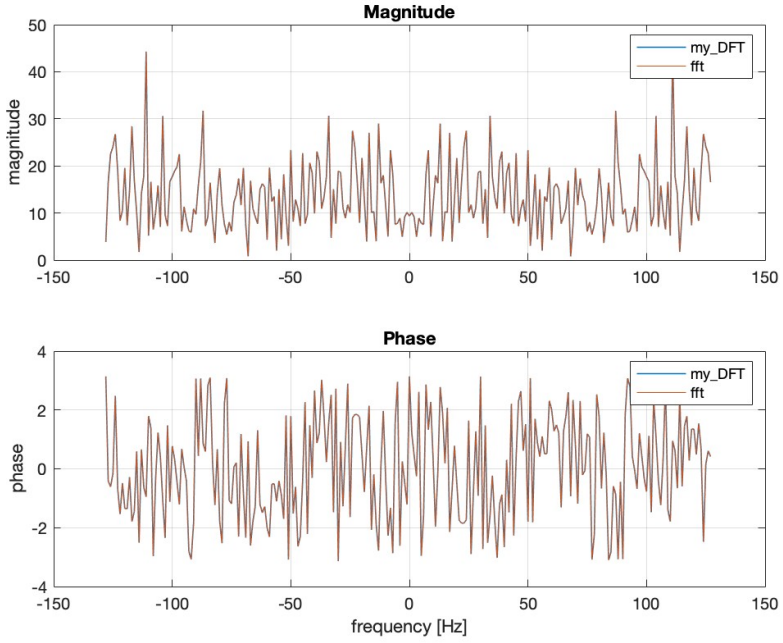

Figure 13: Random sequence DFT

In Figure 13 we can observe that both magnitude and phase of our custom DFT coincide with the built-in Matlab function.

2.3 Exercise 6

For this exercise we implement a Matlab function which computes both the linear and the circular convolution, based on user choice.

2.3.1 Linear convolution

The linear convolution is implemented through matrix multiplication. The matrix we compute will be such that given the input vectors $x_{1\times N}$ and $y_{1\times M}$, the linear convolution z will be given by

$$z = B \cdot y'$$

where B is our matrix.

To compute it, first we zero pad the first vector x in order for it to be of length $N + M - 1$. Then we proceed to create the matrix B where each column is the vector x, each time shifted with the function *circshift* of 1 position.

$$B = \begin{bmatrix} x(1) & 0 & \dots & 0 & 0 \\ x(2) & x(1) & & \vdots & \vdots \\ x(3) & x(2) & \dots & 0 & 0 \\ \vdots & x(3) & \dots & x(1) & 0 \\ x(N-1) & \vdots & \ddots & x(2) & x(1) \\ x(N) & x(N-1) & & \vdots & x(2) \\ 0 & x(N) & \ddots & x(N-2) & \vdots \\ 0 & 0 & \dots & x(N-1) & x(N-2) \\ \vdots & \vdots & & x(N) & x(N-1) \\ 0 & 0 & 0 & \dots & x(N) \end{bmatrix}_{(N+M-1)\times M}$$

In Figure 14 we can see the comparison between the convolution computed with our custom function and the built-in Matlab function *conv*. We can notice that the two result match perfectly.

2.3.2 Circular convolution

The circular convolution is also computed by matrix multiplication. Given the two input sequences $x_{1,N}$ and $y_{1,M}$, the resulting circular convolution sequence will be $z_{1,max(N,M)}$ computed as

$$z = B \cdot y'$$

The matrix B is computed in a similar way to the linear convolution. First we zero-pad both vector in order for them to be of length $N_z \max(N, M)$, then

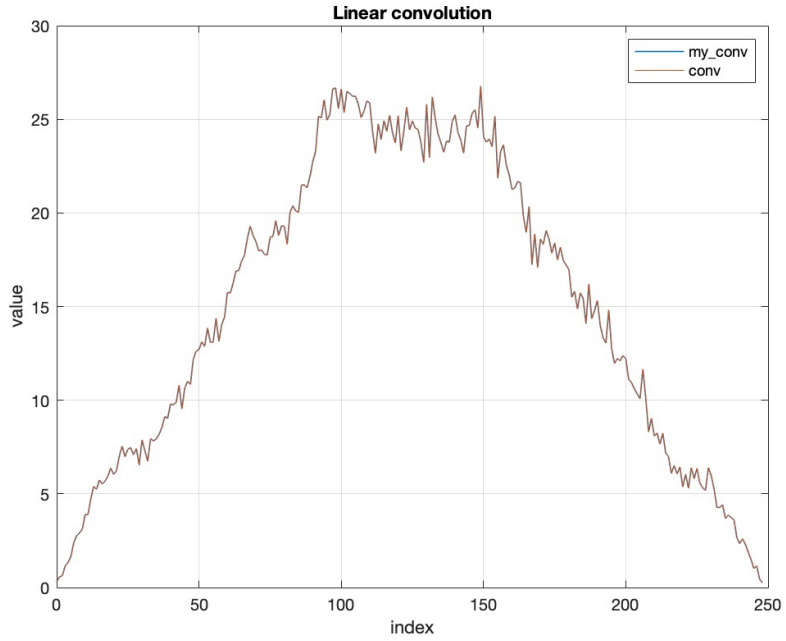

Figure 14: Linear convolution of 2 random sequences

the matrix will be

$$B = \begin{bmatrix} x(1) & x(N_z) & x(N_z-1) & \dots & x(2) \\ x(2) & x(1) & x(N_z) & \dots & x(3) \\ x(3) & x(2) & x(1) & \dots & x(4) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x(N_z-1) & x(N_z-2) & x(N_z-3) & \dots & x(N_z) \\ x(N_z) & x(N_z-1) & x(N_z-2) & \dots & x(1) \end{bmatrix}_{N_z \times N_z}$$

In Figure 15 the circular convolution of the same random sequence is plotted. We can see that it matches the circular convolution computed with the built-in function *cconv*.

2.4 Exercise 7

For this exercise we generate a triangular sequence of $N = 25$ samples by the convolution of two rectangular sequences. Each rectangular sequence must be of length $M = (N + 1)/2$.

The sequence is generated using the custom function *my_conv* and then normalized by M. It is possible to visualize the result in Figure 16

2.4.1 DFT

We now compute the DTFT of this sequence using the *fft* Matlab function. Since N is odd we create the frequency vector bounded in $[-\frac{1}{2} + \frac{1}{2N}, \frac{1}{2} - \frac{1}{2N}]$.

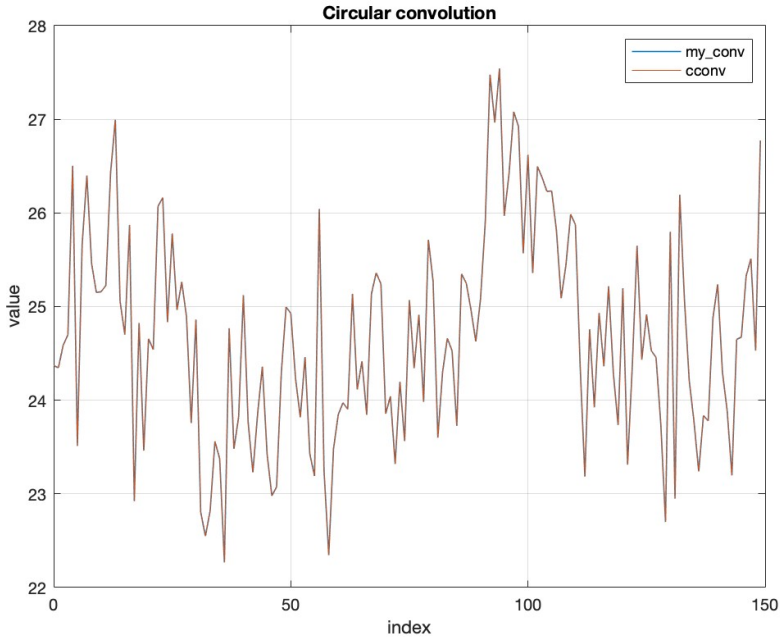

Figure 15: Circular convolution of 2 random sequences

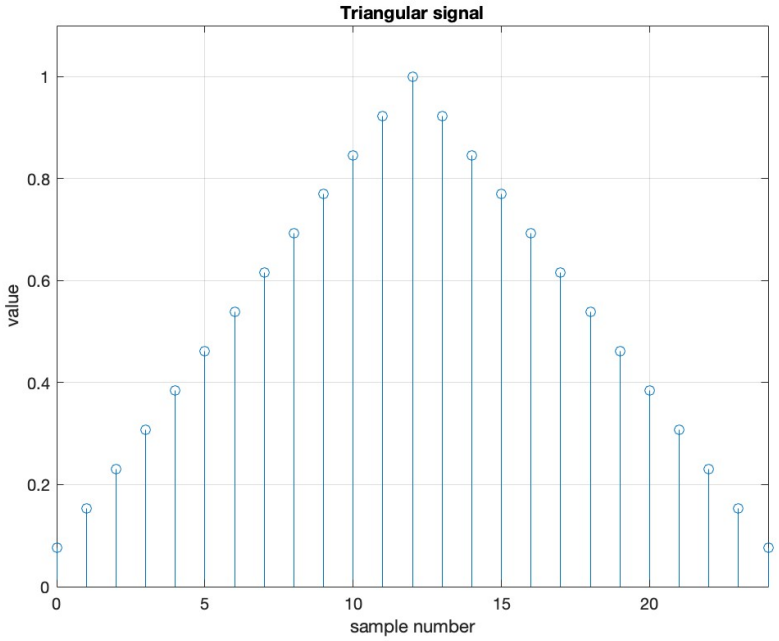

Figure 16: Triangular sequence

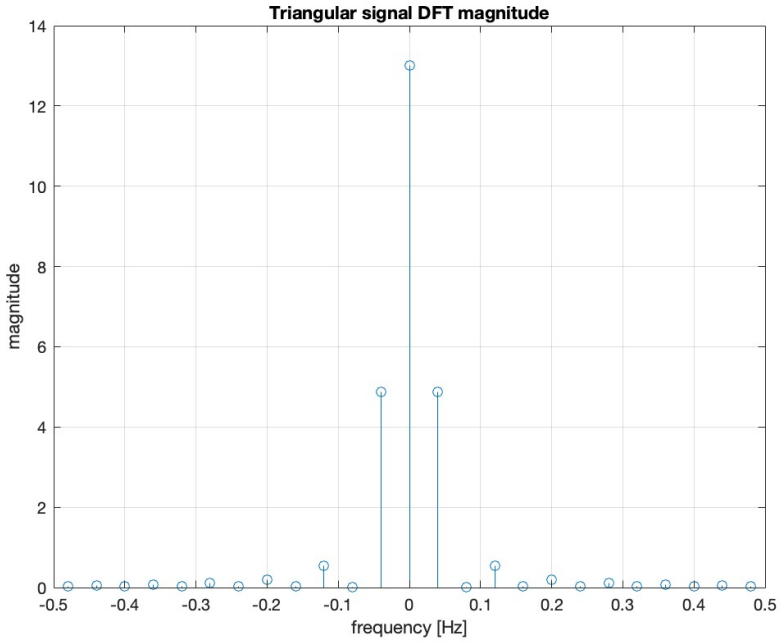

Figure 17: DTFT of triangular sequence

The DFT is plotted in Figure 17

2.4.2 DFT zero-padding

In order to increase the spectral resolution we zero-pad the sequence to get $N = \{32, 64, 128\}$.

We compare the obtain result with the DTFT of the signal defined as

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

In Figure 18 it is possible to see the effect of the zero-padding and how the spectral resolution increases as the number of samples of the sequence increases.

3 FFT and Spectral Analysis of analog signals through DFT

3.1 Exercise 8

Let's consider the function in time domain

$$x(t) = 36te^{-12t}H(t)$$

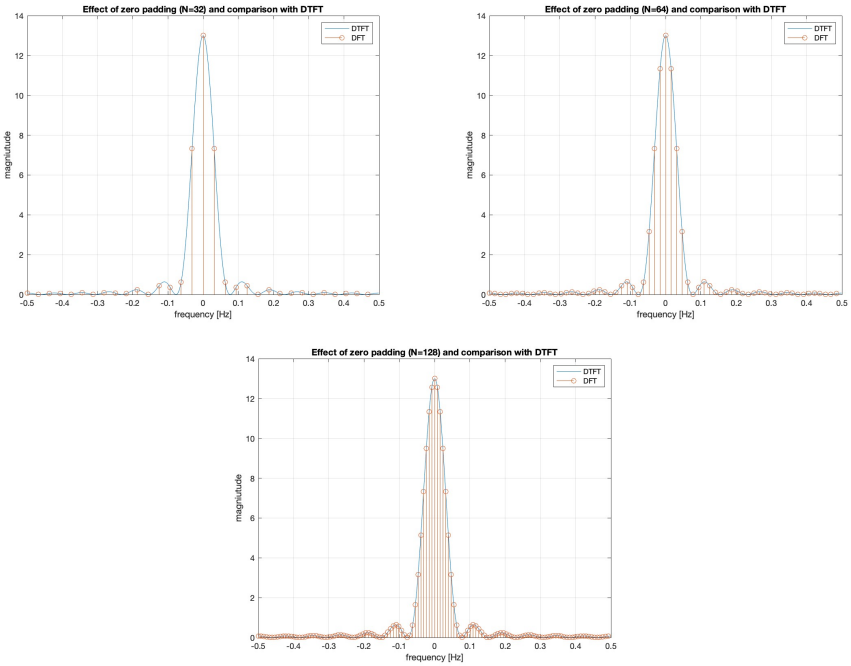

Figure 18: DTFT of triangular sequence zero-padded

3.1.1 Signal Energy

The energy of the signal is

$$\mathcal{E}\{x(t)\} = \int_{-\infty}^{+\infty} |x(t)|^2 \, dt =$$

$$= \int_{0}^{+\infty} 36^2 t^2 e^{-24t} \, dt = \left[-\frac{3}{16} e^{-24t} \right]_0^{+\infty} = \frac{3}{16}$$

we compute the energy integral using Matlab symbolic integration to compare it with the theoretical result. The out come is the same as expected $\mathcal{E}\{x(t)\} = 3/16$.

3.1.2 Signal Fourier Transform

To compute the CTFT we can use the symbolic Fourier transform

$$\mathcal{F}\{x(t)\} = \frac{36}{(12 + j2\pi f)^2}$$

which it can be easily verified is correct from Fourier transform tables.

3.1.3 Bandwidth containing 99.9% of Total Energy

To compute the bandwidth containing the 99.9% of the total energy we define a new symbolic variable B_x in Matlab and solve symbolically the expression

$$\int_{-B_x}^{B_x} |X(f)|^2 \, df = 0.999 \cdot \mathcal{E}\{x(t)\} = 0.999 \cdot \frac{3}{16}$$

$$\frac{9B_x}{4(B_x^2\pi^2 + 36)} + \frac{3\arctan(\pi B_x/6)}{8\pi} = 0.999 \cdot \frac{3}{16}$$

we can now solve the equation for B_x, notice that in this case a symbolic solution cannot be found, thus a numerical approach is used

$$B_x = 14.2507$$

3.1.4 Comparison with Discrete Signal

Let's consider now the discrete signal, considering $f_c = 8B_x$ and T_0 computed as the time in which the signal gets close to 0 ($x(t) < 10^{-3}$).

It is possible to get the number of samples as $N = T_0 f_c$, we round it up to the next power of 2. We have all it is needed to compute the discrete signal.

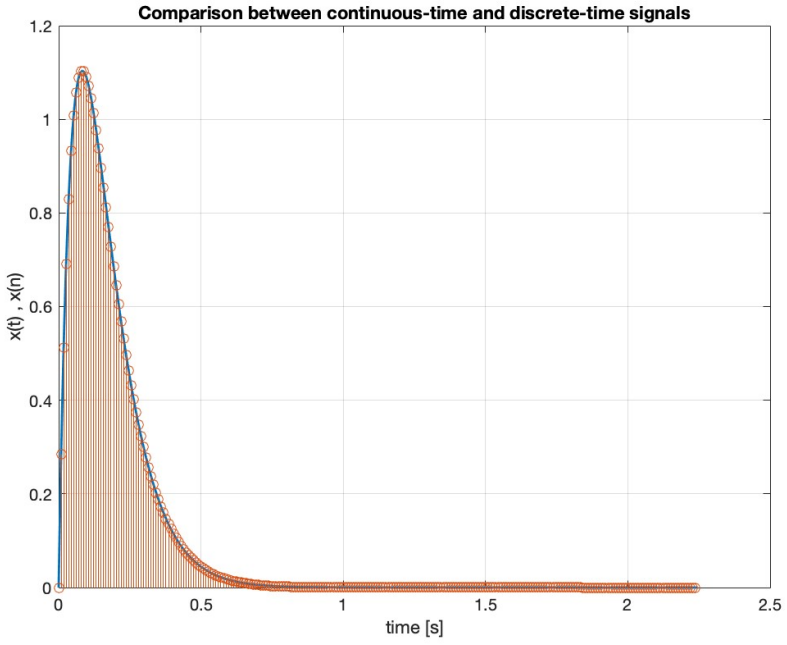

Figure 19: Continuous and Discrete Time Domain Signal

We now compare the Fourier Transform of the signal both continuous and discrete

Δf of the discrete time signal is 0.44534

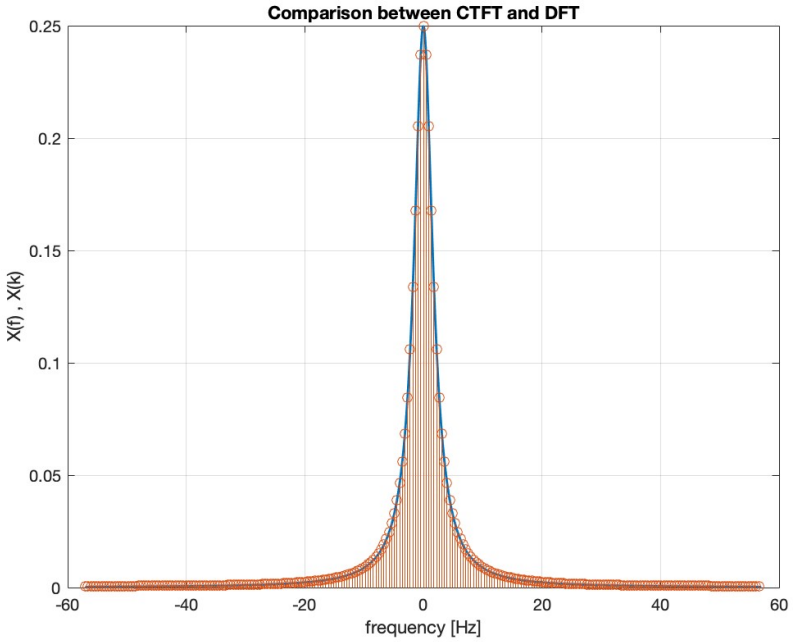

Figure 20: Continuous and Discrete Frequency Domain Signal

If we take a closer look at Figure 20, it can be noticed that as we get further from the zero, the discrete time signal gets higher values than the continuous time.

This is due to the fact that having a discrete signal will result in a periodic signal in frequency domain. Also, since the signal is bounded in time, it will be unlimited in frequency. For this reason the discrete signal will be subject to aliasing. We can visualize it in Figure 21

3.2 Exercise 9

Let's consider the sequence

$$x(n) = 3\cos\left(2\pi\frac{f_0}{f_c}n\right) + 2\cos\left(2\pi\frac{f_1}{f_c}n\right)$$

where $f_0 = 20\text{Hz}$, $f_1 = 16\text{Hz}$, $f_c = 64\text{Hz}$ and $N = 128$.
We can compute T_0 as N/f_c which is 2s and Δf as $1/T0$ resulting 0.5Hz.
Considering the analog signal corresponding to $x(n)$

$$x(t) = [3\cos(2\pi f_0 t) + 2\cos(2\pi f_1 t)] \cdot \text{rect}_{T_0}(t - T_0/2)$$

we can compute the Fourier Transform with the help of Matlab symbolic transform.

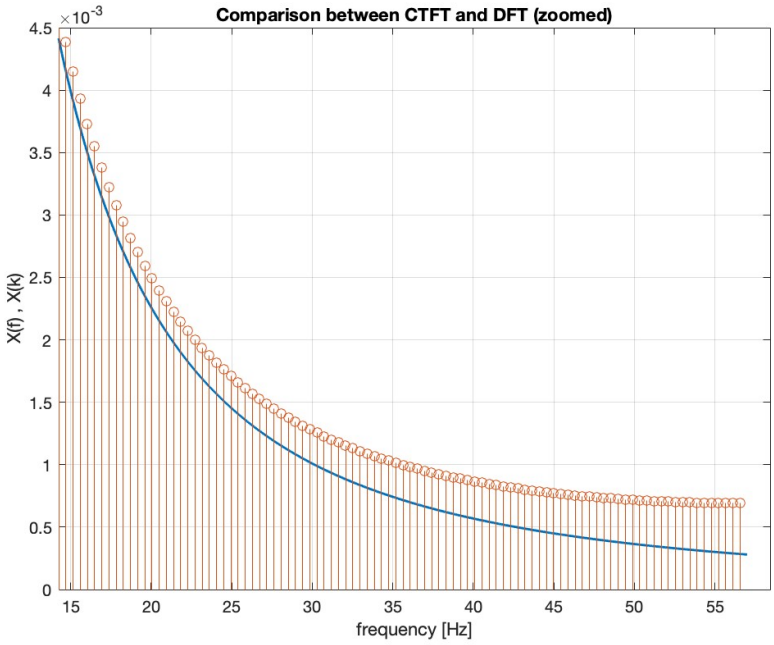

Figure 21: Continuous and Discrete Frequency Domain Signal (Zoomed)

3.2.1 DFT and CTFT comparison

We can compare the DFT of the discrete function computed with the *fft* command and the theoretical CTFT

In Figure 22 it is important to notice that the DFT is always zero, except for the frequencies corresponding to the two sinusoidal functions present in $x(n)$. Though the signal is limited on time, thus we should not get pure deltas in frequency, but rather sinc functions centered at those frequencies.

The reason of our result is that the number of samples of our discrete signal is such that all points correspond exactly with the poles of the sinc functions

3.2.2 Change number of samples

If the number of samples is changed such that $T_0(f_0 - f_1)$ is not an integer, the spectrum changes. In Figure 23 it is represented the spectrum for $N = 130$.

3.2.3 Zero-padding

Now we zero-pad the original signal doubling the number of samples. As a consequence Δf is going to be half of the previous one, so 0.25.

In Figure 24 it can be seen the DTF of the signal zero-padded. The sspectrum has double the resolution of the previuous one. Notice that the shape should not change when we zero-pad. Though, in our case is different since the previous case was a particular case where points corresponded with the sinc poles.

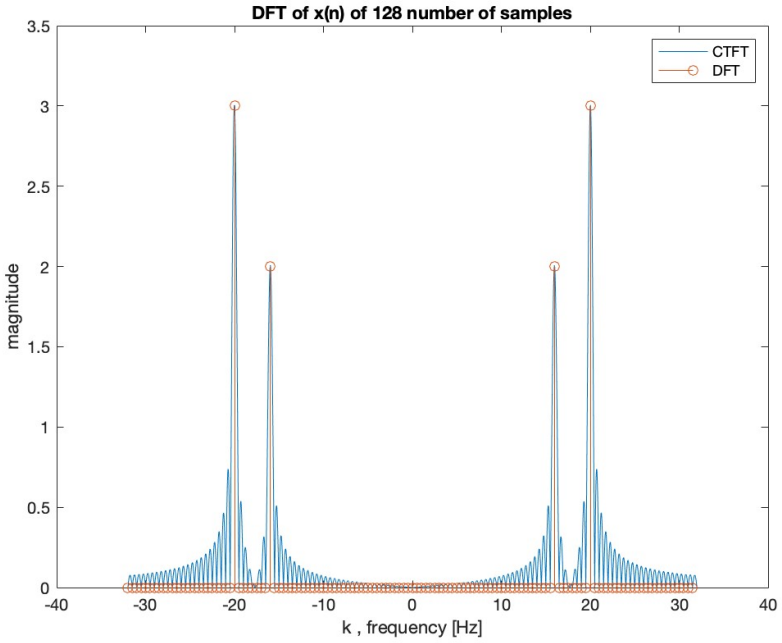

Figure 22: DFT and CTFT of the signal

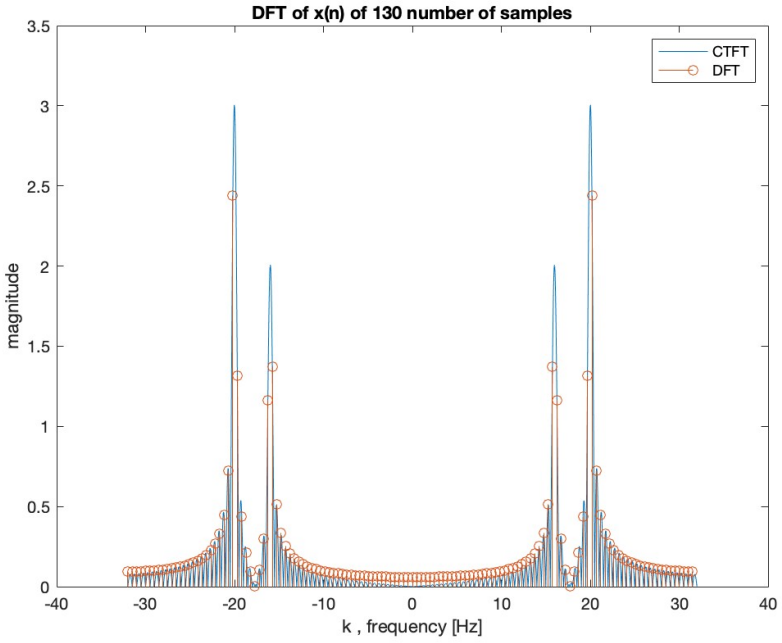

Figure 23: DFT and CTFT of the signal with N=130

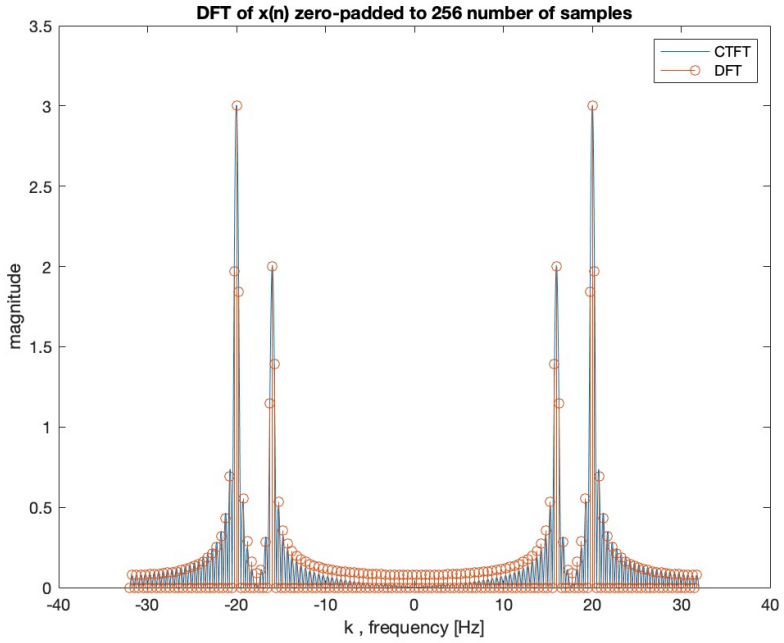

Figure 24: DFT and CTFT of the signal zero-padded

3.3 Exercise 10

For this exercise we consider the bilateral sequence

$$x(n) = 2B_1 \,\mathrm{Sinc}\left(\frac{2B_1}{f_c}n\right) + \frac{B_2}{2}\,\mathrm{Sinc}^2\left(\frac{B_2}{f_c}n\right) + 4B_3\,\mathrm{Sinc}^2\left(\frac{B_3}{f_c}n\right)\cos\left(2\pi\frac{f_0}{f_c}n\right)$$

where $B_1 = 8\text{Hz}$, $B_2 = 3\text{Hz}$, $B_3 = 2\text{Hz}$, $f_0 = 6\text{Hz}$ and $f_c = 32\text{Hz}$. The sequence is made up by $N = 128$ number of samples. As consequence T_0 will be 4s and Δf will be 0.25Hz.

The Fourier transform of its respective analog signal would be

$$Y(f) = \mathrm{rect}_{2B_1}(f) + \frac{1}{2}\,\mathrm{tri}\left(\frac{f}{B_2}\right) + 2\,\mathrm{tri}\left(\frac{f-f_0}{B_3}\right) + 2\,\mathrm{tri}\left(\frac{f+f_0}{B_3}\right)$$

3.3.1 DFT

We now compute the dft of the discrete signal and compare it against the theoretical result. The result can be seen in Figure 25

3.3.2 Zero-padding

We zero-pad the signal with 128 more points so to increase the spectral resolution.

The new value of Δf is 0.125Hz

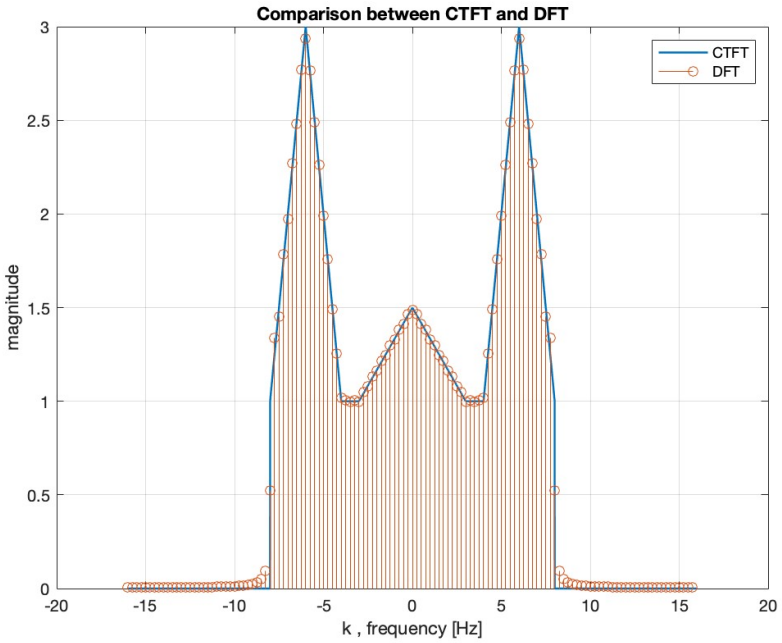

Figure 25: DFT of discrete signal compared to CTFT of analog signal

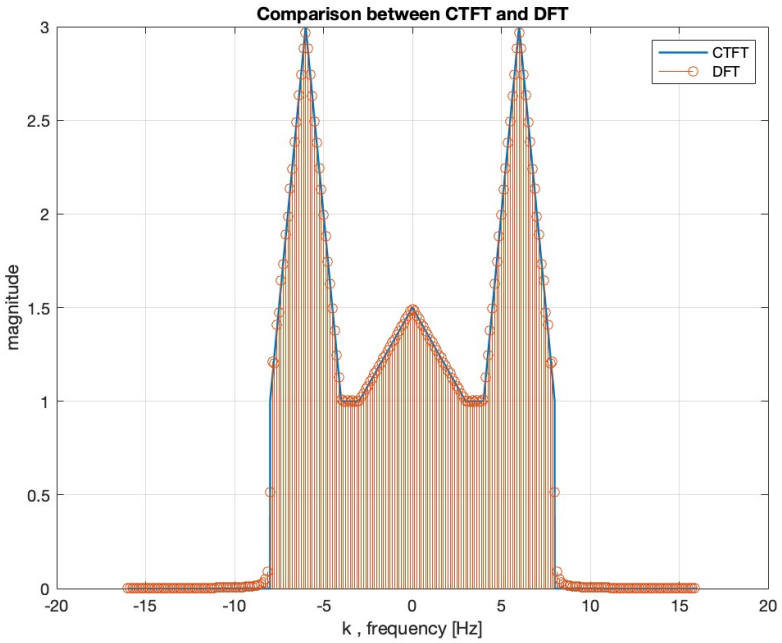

Figure 26: DFT of discrete signal zero-padded compared to CTFT of analog signal

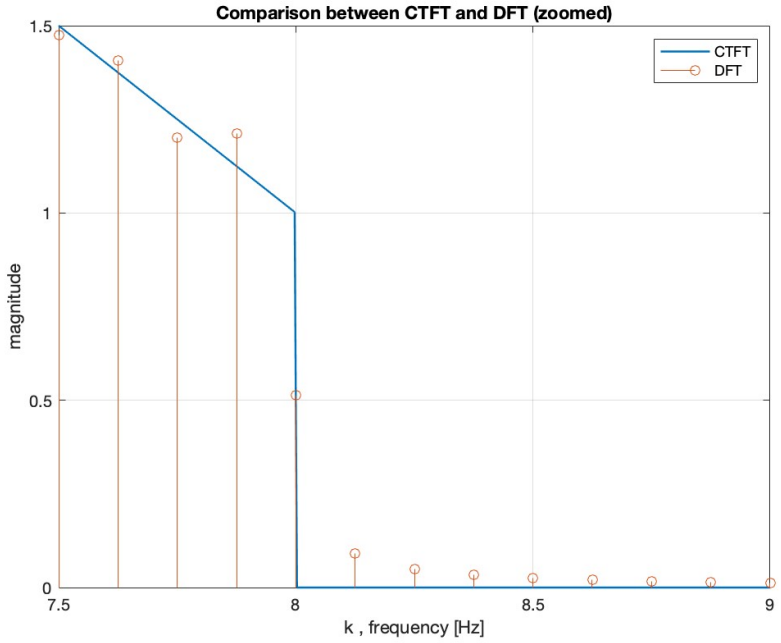

Figure 27: DFT of discrete signal zero-padded compared to CTFT of analog signal zoomed in

In Figure 26 we can observe the effect of zero-padding. Notice that the DFT and the theoretical CTFT does not match precisely. We can have a better look in Figure 27. This is due to the fact that the discrete signal is truncated, thus we would need to multiply the analog signal by a rectangular function. In the frequency domain this can be achieved by convolution with a sinc function

3.3.3 Convolution by Sinc

After computing the convolution in the frequency domain we can now plot again the comparison between the DFT and the new CTFT. It is possible to observe the result in Figure 28.

In Figure 29 we can see closer that the CTFT follow closely the DFT thanks to the applied convolution.

4 Discrete-time correlation functions and random processes

4.1 Exercise 11

In this exercise we generate a WGN signal with the *randn* function, which generates random numbers with a Gaussian distribution. We can see the first 100ms of the the signal in Figure 30. The signal is generated to have zero mean value and variance equal to 25.

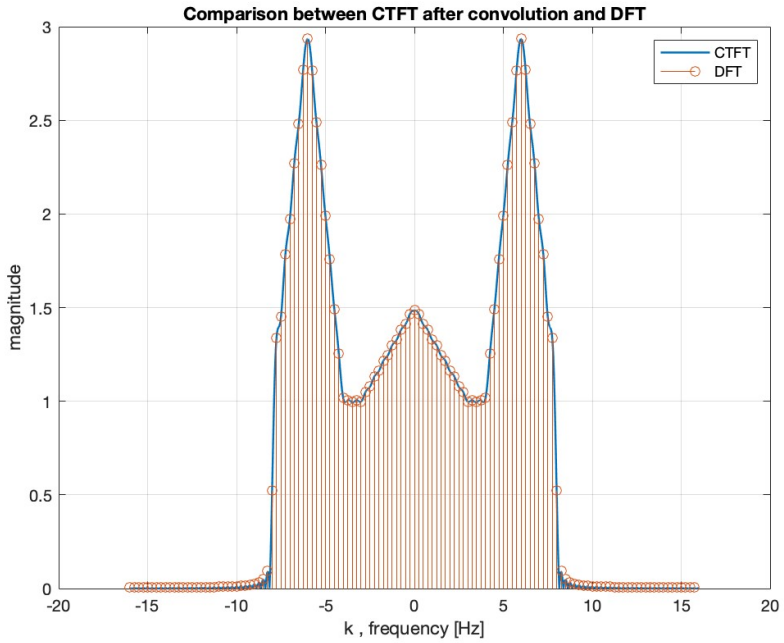

Figure 28: DFT of discrete signal compared to CTFT of analog signal after convolution

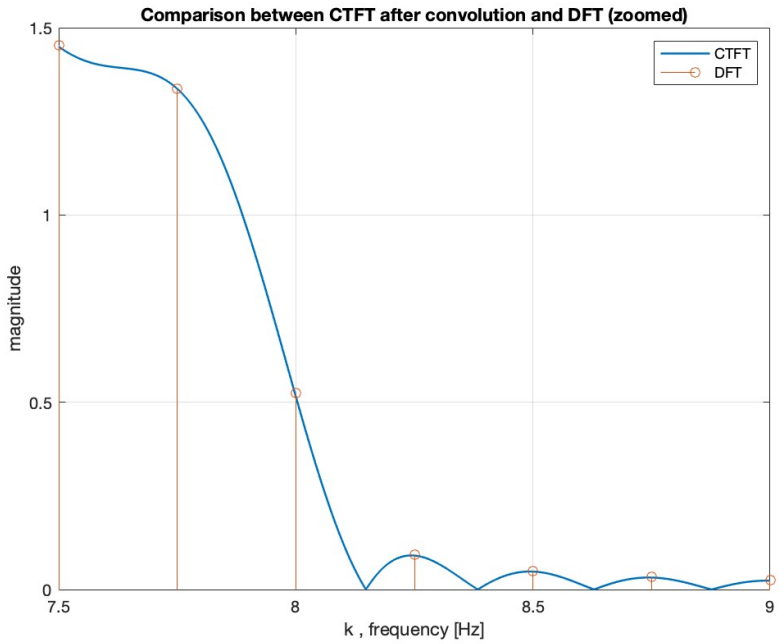

Figure 29: DFT of discrete signal compared to CTFT of analog signal after convolution zoomed in

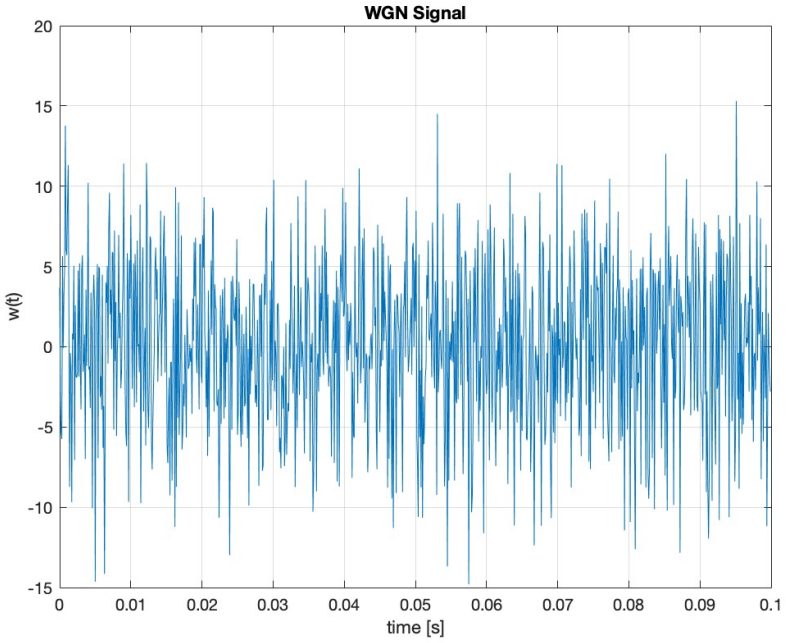

Figure 30: First 100ms of the generated WGN signal

The observed signal has mean value $\mu = 0.022409$, standard deviation $\sigma = 4.9827$ and variance $\sigma^2 = 24.8269$.

It is possible to visualize the PDF of the generated signal in Figure 31. It can be noticed that the shape of the histogram is very close to a Gaussian distribution, plotted in orange, as expected.

The auto-correlation is plotted in Figure 32. As expected is close to zero at each time, except for $t = 0$, where a peak is present. This is typical for a WGN process since all elements generated are independent to each others.

Now we filter the signal with a Low-Pass filter. In Figure 33 the comparison between the original and filtered signal are plotted. The filtered signal is smoother and lower in amplitude.

In Figure 34 instead is plotted the PDF of the filtered signal.

It is interesting to see the effect of the filtering in the auto-correlation. The comparison is plotted in Figure 35. The filtered signal auto-correlation has a lower peak at the origin, and goes to zero smoother with respect to the original signal. This indicates that there is a small dependency on different samples, this is due to the fact that the filter adds some memory in the systems, creating a dependency on close samples.

4.2 Exercise 12

For this exercise we generate a signal of 10s. This signal is zero except for a 4s interval containing a random process. This process is made up by rectangular

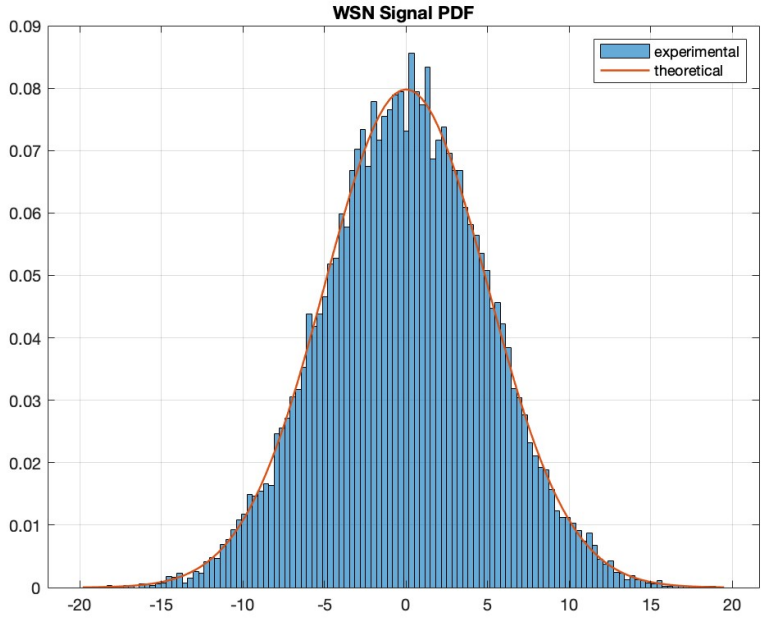

Figure 31: PDF of the generated WGN signal

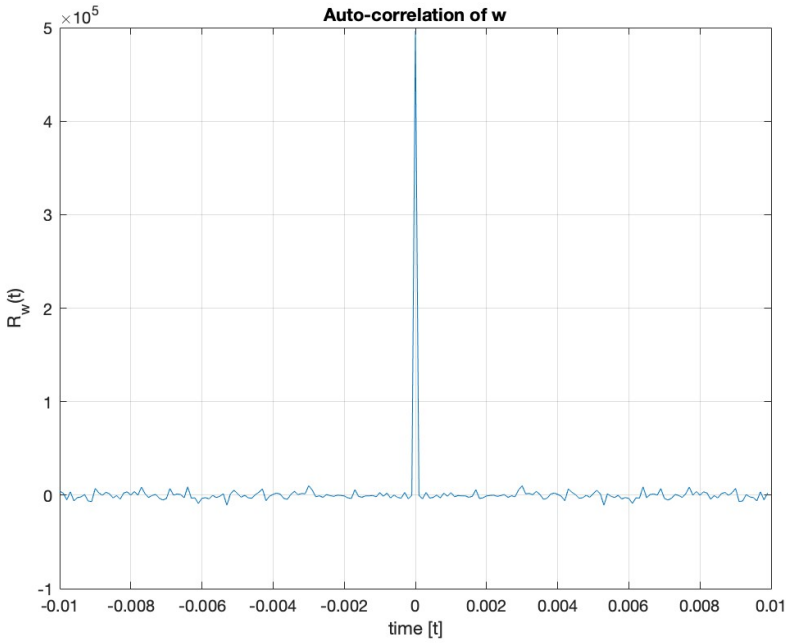

Figure 32: Auto-correlation of the generated WGN signal

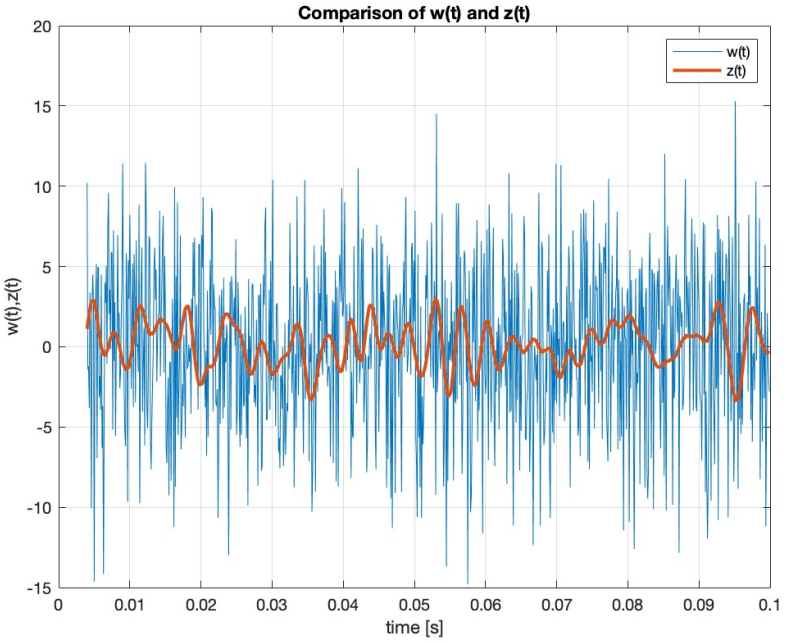

Figure 33: Filtered WGN signal

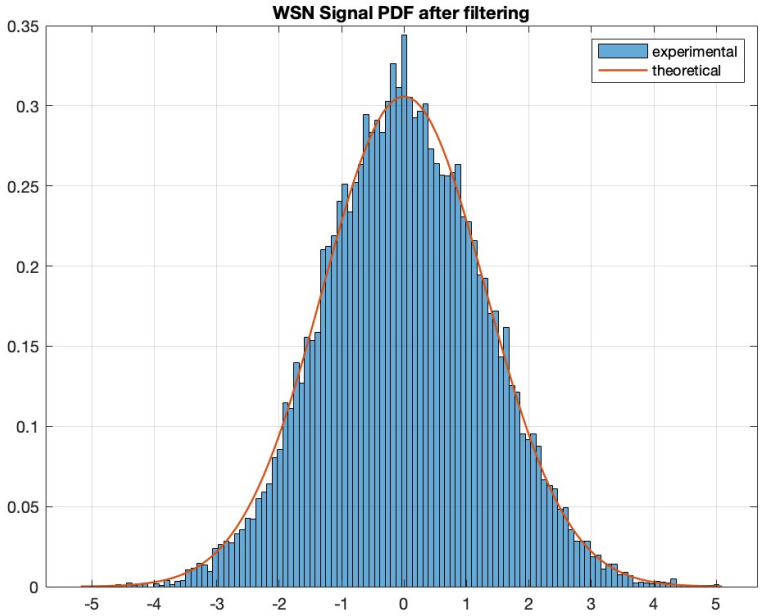

Figure 34: PDF of the filtered WGN signal

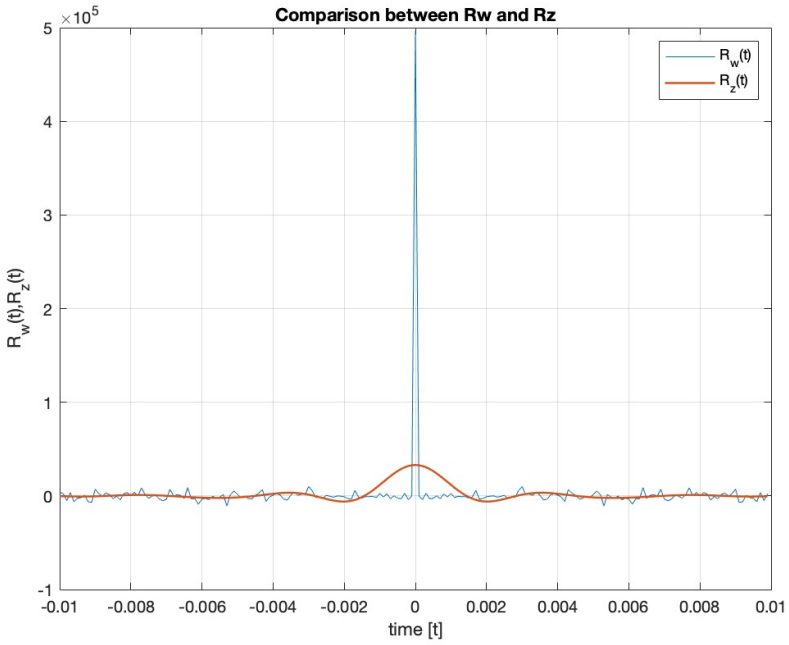

Figure 35: Auto-correlation of the filtered WGN signal

ports 2ms wide that take values $\{-1, 1\}$.

This interval has a random delay from the start of the simulation.

After generating the signal a WGN process is added with zero mean value and variance equal to 25.

In Figure 36 is plotted the generated signal on the top. On the bottom the signal with noise is plotted. It is impossible to distinguish the original signal since the noise completely covers it.

It is possible to estimate the delay by computing the cross-correlation between the noisy signal and the 4s random process. The start of the random interval will coincide with the peak value.

In order to do that we create a custom function *my_ corr*. This function will accept two vectors as input arguments and returns the cross-correlation between them. This function use the same method that the *my_ conv* used to compute the convolution, but with a vector flipped. Reference Exercise 6 (2.3.1) for the function implementation.

In Figure 37 the cross correlation is plotted. The peak is clearly visible in the figure. The estimated delay is 0.84s which is exactly the delay of the signal.

If we simulate again with a WSG signal with variance equal to 50, the delay is still correctly estimated.

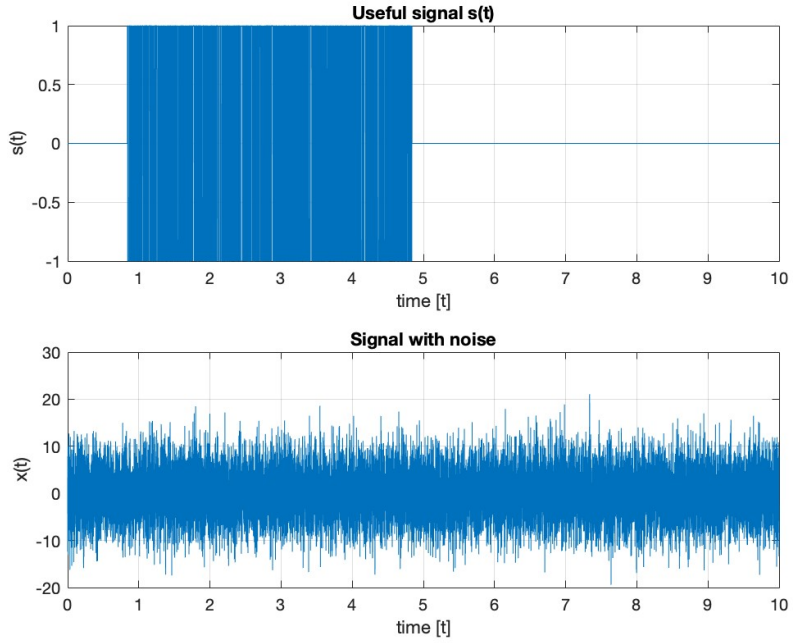

Figure 36: Generated signal with and without noise added

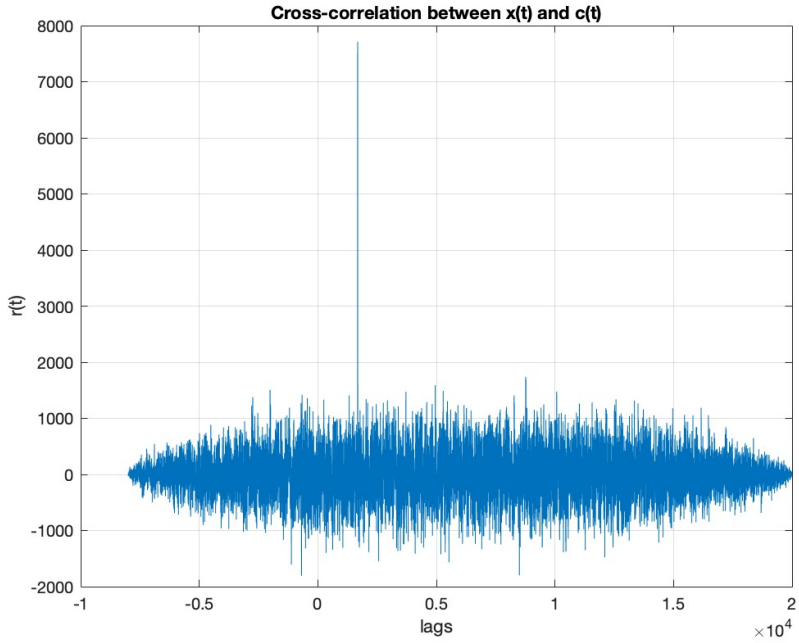

Figure 37: Cross-correlation between signal with noise and random signal

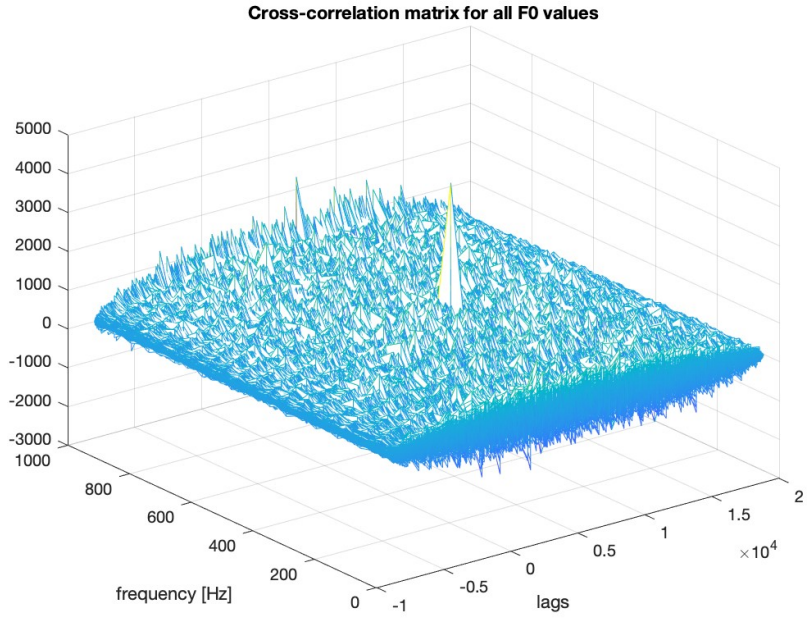

Figure 38: Cross-correlation matrix between signal with noise and random signal

4.3 Exercise 13

For this exercise we will use the same signal of Exercise 12 (4.2). This time the random part of the signal will be modulated on the frequency f_0 by multiplying the random signal by $\cos(2\pi f_0 t)$. f_0 is computed randomly as $f_0 = p \cdot 50\text{Hz}$, with p an integer from 1 to 20.

This time we need to estimate 2 values, the delay and the modulation frequency. For this reason we compute the cross correlation, same way as in exercise 12 (4.2), for each value of p.

We can visualize it with a 3D plot in Figure 38

The estimated values can be derived by the 2D coordinates of the peak value of the cross-correlation matrix. The estimated delay is 3.52 and the estimated frequency is 500. The values are accurate.

Simulating again with WSG noise with variance equals to 50 the delay and frequency can still be computed, though the difference between the peak and the rest of the function is less relevant with respect to the previous exercise.

5 Fundamentals of spectral estimation

5.1 Exercise 14

Let's consider the following signal

$$x(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t) + \cos(2\pi f_3 t) + W(t)$$

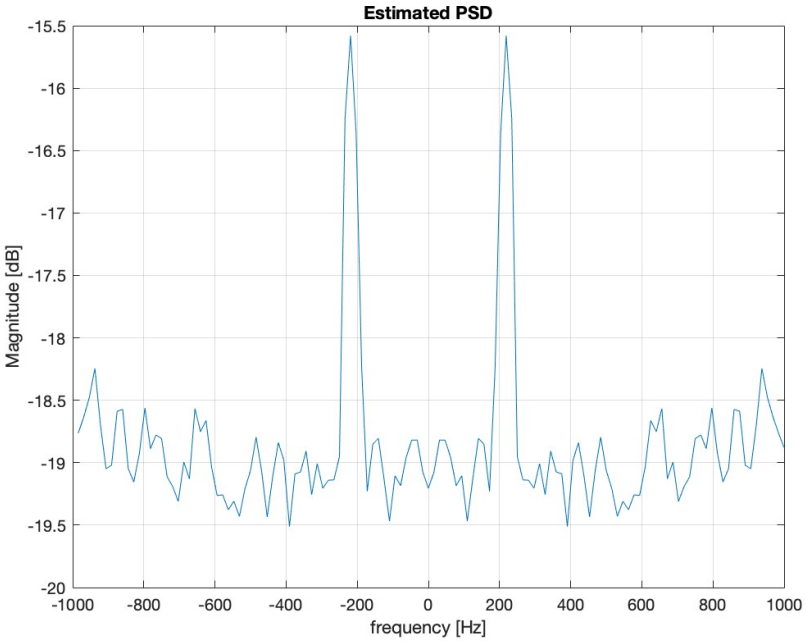

Figure 39: Estimated PSD

where $f_1 = 200$ Hz, $f_2 = 220$Hz, $f_3 = 230$Hz and $W(t)$ is a zero mean WGN signal with variance $\sigma^2 = 25$.

5.1.1 PSD estimation

We estimate the power spectral density $S(f)$ with the use of the *pwelch* Matlab function. We set as parameters 128 as spectral resolution, hamming windowing and no window overlap.

In Figure 39 the estimated PSD is plotted. As expected we see a peak around 200Hz, though we had to see three peaks, this means that the periodogram is not that accurate.

5.1.2 Optimized periodogram

We get a better result by setting optimized the values for the *pwelch*. We sat the number of samples per segment to 1000 and 500 samples of overlapping.

In Figure 40 we can see the result plotted. Now the three deltas are clearly distinguishable.

5.1.3 Periodogram comparison

We now compute the periodogram with three different techniques. We will always use the *pwelch* function changing the parameters each time. The parameters to change are: the number of samples per segment $NFFT$, the window and the number of overlapped samples $n\_overlap$.

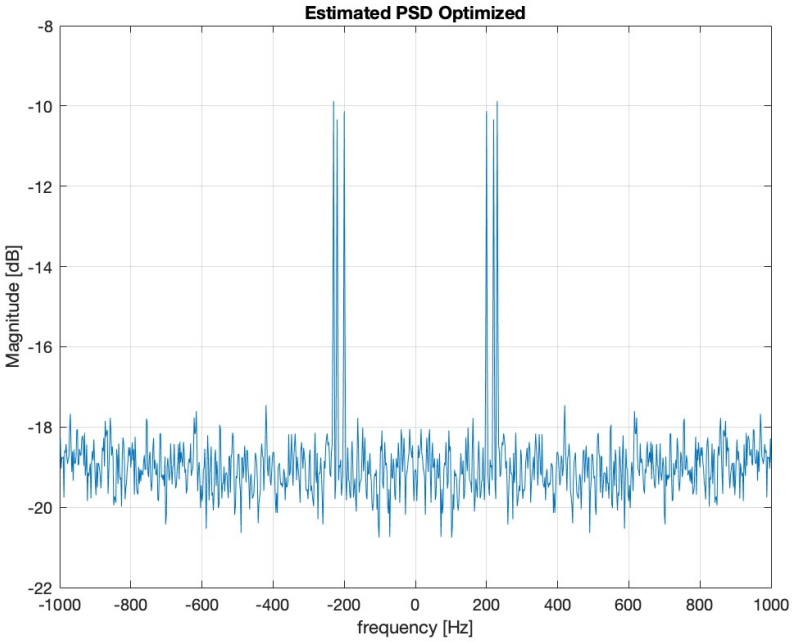

Figure 40: Estimated PSD (optimized)

The three periodograms are:

- simple: $NFFT$ = number of samples of the signal, rectangular window, $n\_overlap = 0$
- bartlett: $NFFT = \frac{\text{number of samples}}{\text{number of segments}}$, rectangular window, $n\_overlap = 0$
- welch: $NFFT = \frac{2 \cdot \text{number of samples}}{\text{number of segments}+1}$, hamming window, $n\_overlap = \frac{NFFT}{2}$

Figure 41 shows the comparison of the three different methods. We can notice how the simple periodogram accentuate more the peaks at the three frequencies, but is less accurate having an high variance, while the bartlett and the welch are pretty similar and have a smaller variance, which makes them more accurate.

5.1.4 Auto-correlation estimation

The auto-correlation is estimated in two ways with the Matlab function *xcorr* setting the parameter *SCALEOPT* to either 'biased' or 'unbiased'.

In Figure 42 we can see how setting the *SCALEOPT* to 'biased' makes the estine less accurate at the origin but keeps the accuracy as we get far from the origin, while setting the parameter to 'unbiased' makes the estimation very accurate at the origin, but unreliable as the extremes.

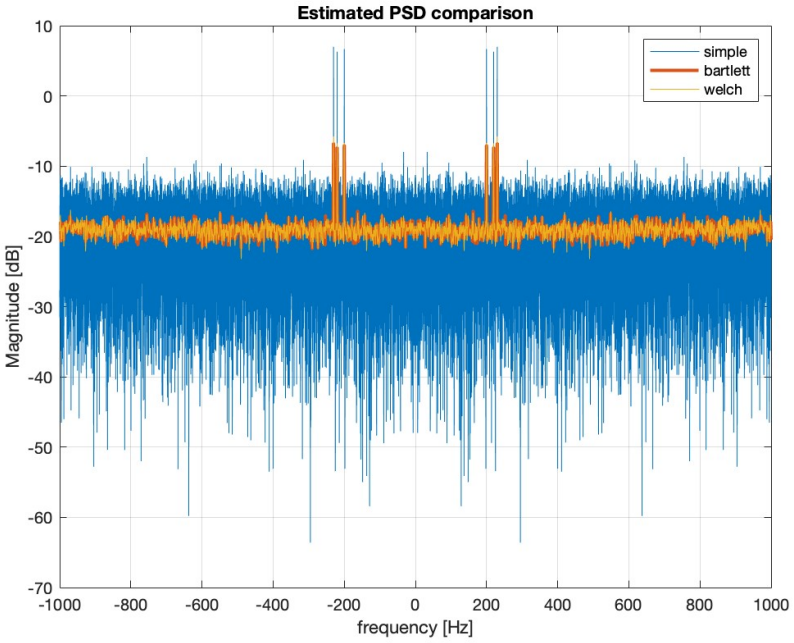

Figure 41: Estimated PSD comparison

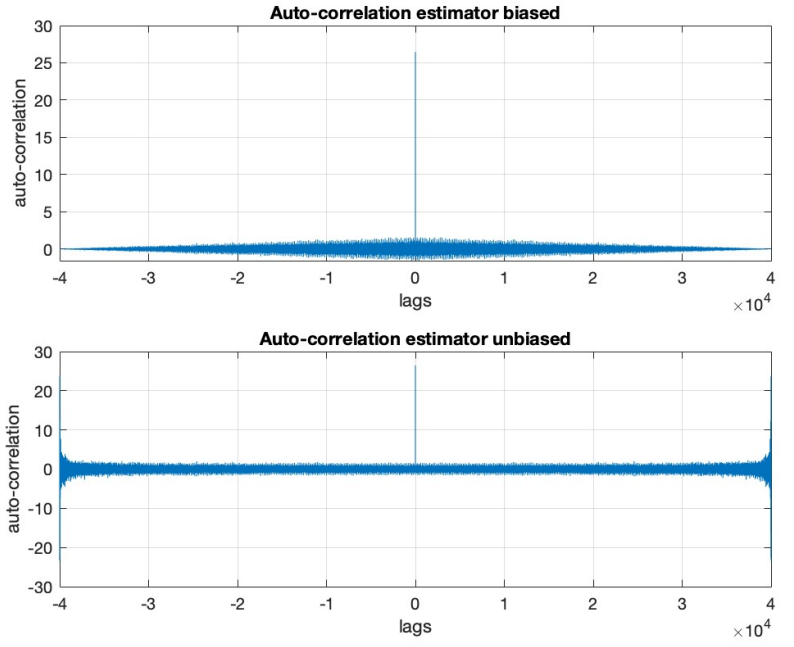

Figure 42: Auto-correlation estimation

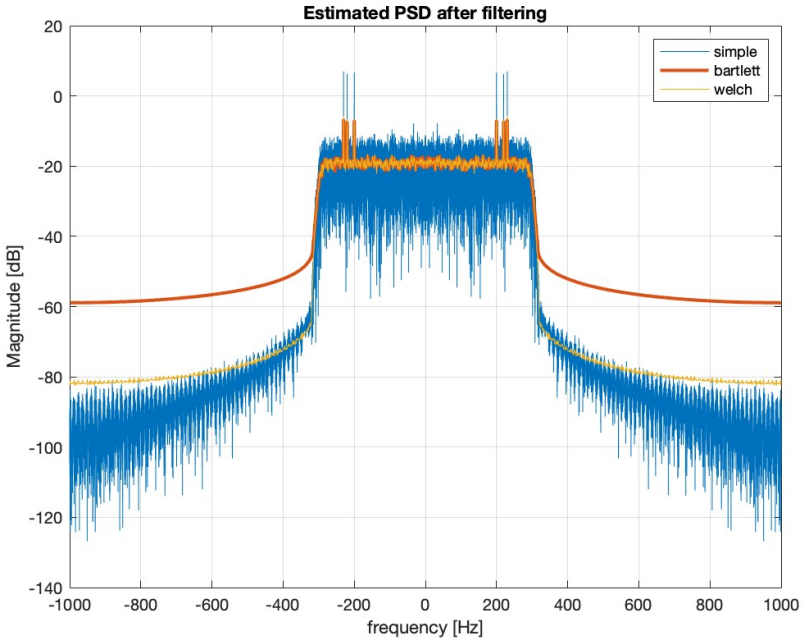

Figure 43: PSD estimation comparison after filtering

5.1.5 Filtering the signal

We apply a Low-Pass filter to our signal and recompute the previous PSD estimations with the three different methods.

In Figure 43 we can see how after the filtering the higer frequencies drop down to -80dB very quickly at about 300Hz.

In Figure 44 we can see the PSD estimated with the welch method but with three different windowing techniques, rectangular, hamming and hann. The difference are clear in the higher frequencies. The hann window is the one that causes a greater drop, than the hamming, than the rectangular. This is what we expected from theory.

5.2 Exercise 15

For this exercise we create a custom function *my_bartlett* that estimate the spectral density of a vector computing the Bartlett periodogram. It accepts 3 input parameters, a vector, the number of samples N and the number of segments N.

5.2.1 Implementation

First the function checks that M, is not greater than N, otherwise an error is thrown. Then a zero-padding is applied to make the length of the vector a multiple of M.

We can now compute the segment size D as N/M.

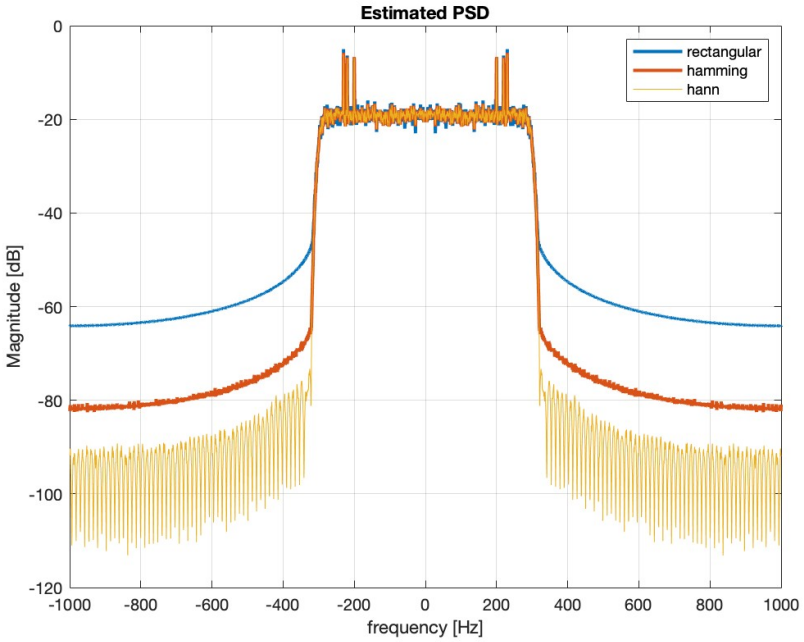

Figure 44: PSD estimation comparison after filtering (welch method)

A matrix $M \times D$ is initialized and populated with a for loop with M periodogram computed for each segment.

First we compute the absolute square value of each element and then the mean value is computed.

5.2.2 Testing

Now we test the custom function by computing the Barlett periodogram first with the use of *my_ Bartlett* and later with the use of *pwelch* Matlab function. The simulation is carried out by generating a random sequence of 1000 samples and setting M, the number of segments, to 50.

In Figure 45 we can observe the results, the two PSD perfectly match. Though, in this case N is a multiple of M, if we make this not the case, the simulation presents some slight differences. This might be due to a difference in handling the values of M and N being not multiples. The custom function applies a zero-padding, the *pwelch* function might work in a different way.

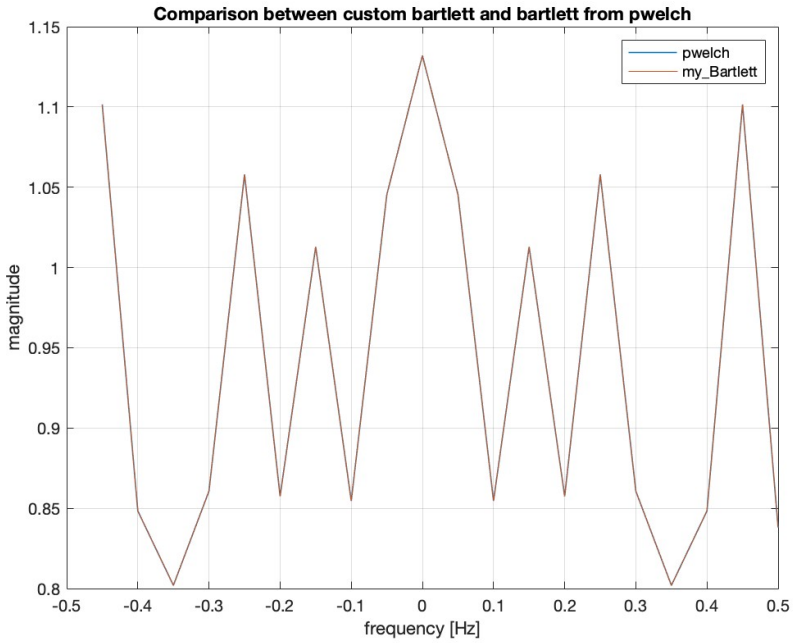

Figure 45: Estimated PSD comparison