University of Brasília

Electrical Engineering Department

Topics in Biomedical Engineering

Exercise List 4

Authors:

Caio Luiz Candeias Flôres 190134283

Felipe Carneiro da Motta 200017616

João Pedro Daher Aranha 190109742

Brasília

August 6, 2022

Contents

1 Exercises

1.1 Exercise 3.5:

Construct the sawtooth wave shown below and use the MATLAB fft routine to find the frequency spectrum of this waveform. Then reconstruct the square wave using the first 24 components using Equation 3.15. Do not plot the spectrum, but plot the original and reconstructed waveform superimposed. Make fs = 1024 and N = 1024 to represent the 1-s periodic waveform. [Hint: The MATLAB fft routine does no scaling; so, you will have to scale the output to get the correct reconstructed waveform. Also, remember that the first element of the complex vector produced by the fft routine is the DC component (that will be 0.0 for this waveform) and the first complex sinusoidal coefficient is in the second element.]

The MATLAB's code:

```
clc; close all; clear all;

fs = 1024; % sampling frequency
N = 1024; % number of points
A = 0.5; % amplitude

%% sawtooth wave

% sawtooth time domain
f1 = 2; % freq of the signal
t = (0:N-1)/fs; % time axis
s1 = A*sawtooth(2*pi*f1*t); % signal

% signal reconstructed
% fft
c = 24; % number of points to reconstruction
sf = (2/N)*fft(s1); % fft
S_mag = abs(sf); % fft mag
f = (0:N-1)*(fs/N); % frequency axis
S_phase = unwrap(angle(sf)); % fft phase

r1 = zeros(1,N); % array of zeros
r1_dc = S_mag(1)/2; % dc value
```

```
r1 = r1_dc + r1;  % add dc value

for m = 2:c % 2:c because S_mag(1) is a dc term
    r1 = r1 + S_mag(m)*cos(2*pi*f(m)*t + S_phase(m)); %
        Eq. 3.15
end

plot(t, s1, 'k', 'linewidth', 1, 'DisplayName', 'Signal
    ');
hold on;
plot(t, r1, 'b', 'linewidth', 1, 'DisplayName', '
    Reconstructed Signal');
xlabel('Time (s)');
ylabel('$s(t)$', 'Interpreter', 'latex');

legend('Orientation','horizontal','Box','on','Location'
    ,'southoutside');
title('Signal reconstruction of $s(t)$ w/ 24 components
    ', 'Interpreter', 'latex');
grid on;

saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

According to the figure 1, it's possible to visualize that the reconstructed sawtooth wave follows the shape of the original sawtooth wave with some oscillations mainly close of abrupt transitions. This phenomenon is called of "Gibbs Phenomenon" and it occurs due the union of sinusoids with different frequencies.

The reconstruction with the first 24 components has already generated a good rebuil of the waveform by the Equation 1. In discrete terms, more specificly in MATLAB operations, this equation begins in the 2^{nd} term because the first is a DC term. Evidently, according to the increase of terms used, the rebuilt wave gets closer to the original wave still with the glitch effect in abrupt transitions. In this case, 24 components was a good approach in terms of visualization.

$$x(t) = \frac{a_0}{2} + \sum_{m=2}^{\infty} X_{mag}(m)cos(2\pi ft + X_{phase}(m)) \tag{1}$$

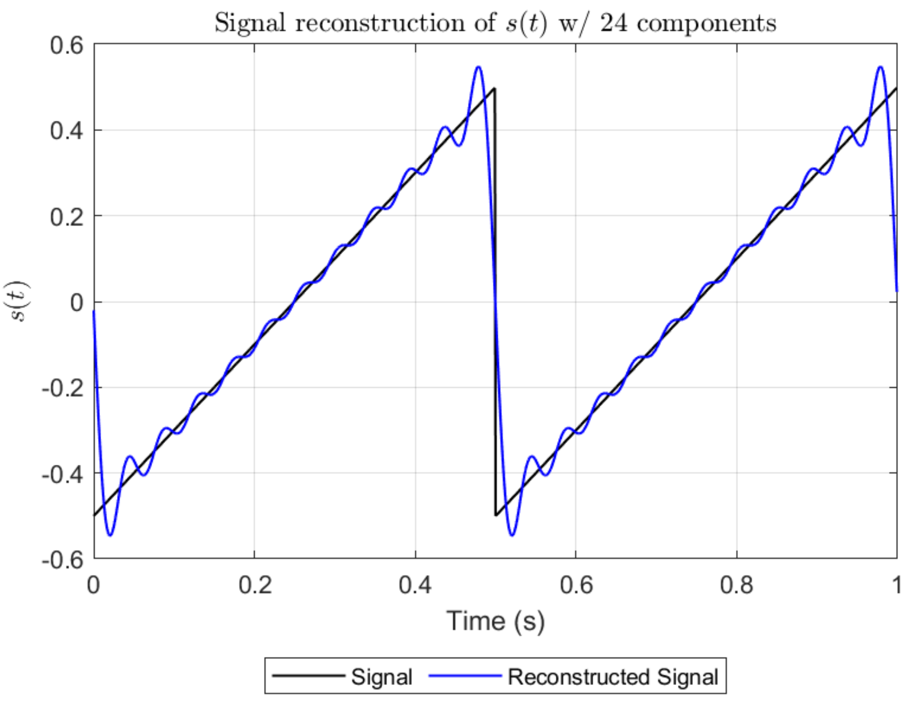

Figure 1: Original Sawtooth Wave and Rebuilt Sawtooth Wave.

1.2 Exercise 3.11:

The data file pulses.mat contains three signals: x1, x2, and x3. These signals are all 1.0 s in length and were sampled at 500 Hz. Plot the three signals and show that each signal contains a single 40-ms pulse, but at three different delays: 0, 100, and 200 ms. Calculate and plot the spectra for the three signals superimposed on a single magnitude and single phase plot. Plot only the first 20 points as discrete points, plus the DC term, using a different color for each signal's spectra. Apply the unwrap routine to the phase data and plot in deg. Note that the three magnitude plots are identical and while the phase plots are all straight lines, they have radi- cally different slopes.

The MATLAB's code:

```
close all
load pulses.mat
N=500;              %signals' lenght
```

```
fs=500;           %Sampling frequency
t=(0:N-1)/fs;     %time vector
subplot(3,1,1)    %Plotting the raw signals
plot(t,x1)
title('Signal x1')
xlabel('Time(s)')
ylabel('Amplitude')

subplot(3,1,2)
plot(t,x2)
title('Signal x2')
xlabel('Time(s)')
ylabel('Amplitude')

subplot(3,1,3)
plot(t,x3)
title('Signal x3')
xlabel('Time(s)')
ylabel('Amplitude')

saveas(gcf, 'ex311_1'); % save image

X1=fft(x1); % The signal's FFT
X2=fft(x2);
X3=fft(x3);

X1_mag = (2/N)*abs(X1); % Compute magnitude spectrum
X2_mag = (2/N)*abs(X2);
X3_mag = (2/N)*abs(X3);
Phase1 = unwrap(angle(X1)); % Phase spectrum unwraped
Phase1 = Phase1*360/(2*pi); % Convert phase to deg
Phase2 = unwrap(angle(X2));
Phase2 = Phase2*360/(2*pi);
Phase3 = unwrap(angle(X3));
Phase3 = Phase3*360/(2*pi);

f=(0:N-1)*fs/N; % Frequency vector
figure;
subplot(2,1,1)
plot(f(1:N/2),X1_mag(1:N/2)); hold on;
plot(f(1:N/2),X2_mag(1:N/2),'k'); hold on;
plot(f(1:N/2),X3_mag(1:N/2),'r');
legend('X1','X2','X3')
title('Magnitude')
xlabel('Frequency (Hz)');
ylabel('Amplitude');

subplot(2,1,2)
plot(f(1:N/2),Phase1(1:N/2)); hold on;
plot(f(1:N/2),Phase2(1:N/2),'k'); hold on;
plot(f(1:N/2),Phase3(1:N/2),'r');
legend('X1','X2','X3')
title('Phase in degrees')
xlabel('Frequency (Hz)');
ylabel('Angle (deg)');

saveas(gcf, 'ex311_2'); % save image
```

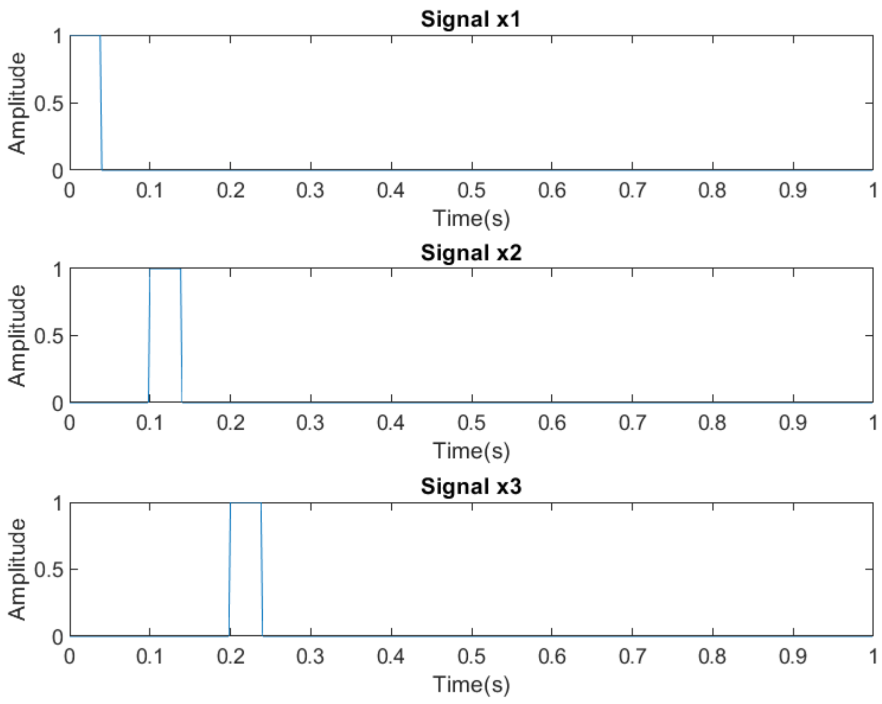

Figure 2: Raw Signals.

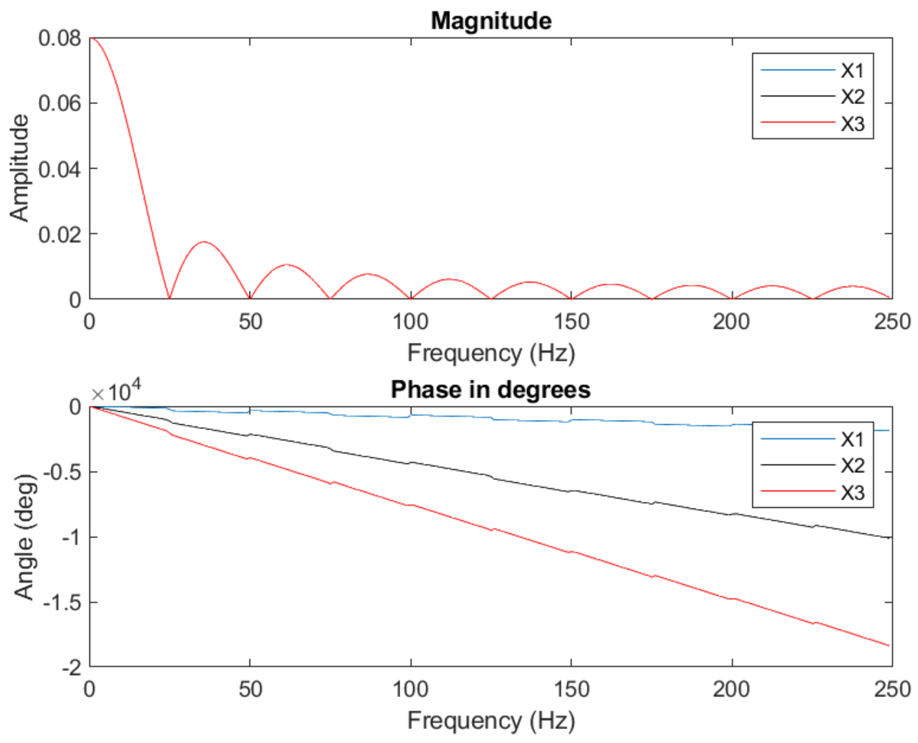

Figure 3: Magnitude and Phase.

Although the phase spectrum is different for the signals, the magnitude spectrum of the signals is the same, that is why we can only see the red line at the figure, that indicates the X3 spectrum, it was the last to be plotted, so it got over the other colors.

1.3 Exercise 3.14:

This problem demonstrates aliasing. Generate a 512-point waveform consisting of two sinusoids at 200 and 400 Hz. Assume fs = 1 kHz. Generate another waveform containing frequencies at 200 and 900 Hz. Take the FT of both waveforms and plot the magnitude of the spectrum up to fs

/2. Plot the two spectra superimposed, but plot the second spectrum as dashed and in a different color to highlight the additional peak due to aliasing at 100 Hz. [Hint: To generate the sine waves, first construct a time vector, t, then generate the signal using x = sin(2*pi*f1*t) + sin(2*pi*f2*t) where f1 = 200 for both signals, whereas f2 = 400 for one waveform and f2 = 900 for the other.]

The MATLAB's code:

```
close all; clear all; clc;
% Defining the time vector with 512 points
N = 512; % number of points
fs = 1000; % sampling frequency
Ts = 1/fs; %sampling period
t1 = (0:N-1)*Ts; % time vector
f = (0:N/2-1)*fs/N; % frequency vector

f1 = 200; %frequency 1
f2 = 400; %frequency 2
f3 = 900; %frequency 3

x1 = sin(2*pi*f1*t1)+sin(2*pi*f2*t1); %signal 1
x2 = sin(2*pi*f1*t1)+sin(2*pi*f3*t1); %signal 2

X1_mag = 2*abs(fft(x1))/N; %magnitude spectrum of signal 1
X2_mag = 2*abs(fft(x2))/N; %magnitude spectrum of signal 2

%plot
hold on;
plot(f(1:N/2),X1_mag(1:N/2),'b','DisplayName','200 Hz and 400 Hz waves');
plot(f(1:N/2),X2_mag(1:N/2),'r--','DisplayName','200 Hz and 900 Hz waves');

title('Demonstration of aliasing');
ylabel('Magnitude');
xlabel('Frequency (Hz)');
legend('Orientation','vertical','Box','on','Location','southoutside');
grid()
saveas(gcf,'ex3_14.png');
```

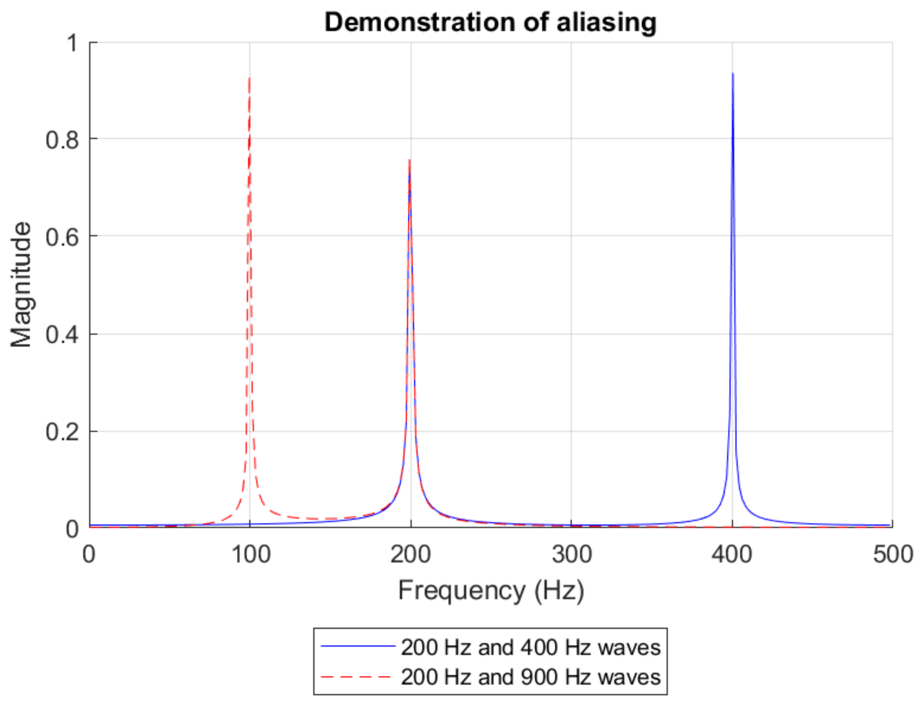

Figure 4: Magnitude of 200 Hz (blue) and 400 Hz and magnitude of 200Hz and 900 Hz(red)

The sampling frequency of 1000 Hz is not high enough in order to identify 900 Hz frequencies. It can be noticed that the algorithm could identify correctly the frequencies of the second signal. The fact that sampling frequency is lower than the Nyquist frequency justifies this phenomenon.

1.4 Exercise 3.17:

The file short.mat contains a very short signal of 32 samples. Plot the magnitude spectrum as discrete points obtained with and without zero padding. Zero pad out to a total of 256 points. Note the interpolation provided by zero padding.

The MATLAB's code:

```
% exercise 3.17 - Semmlow
clc; close all; clear all;
```

```
load short.mat % variable x -¿ 32 points
fs = 1; % sampling frequency
N = length(x); % number of points
f = (0:N-1)*(fs/N); % frequency axis without zero
    padding

N2 = 256; % number points 0 (zero padding)
f2 = (0:N2-1)*(fs/N2); % frequency axis w/ zero padding

X_fft = fft(x); % fft
X_mag = (2/N)*abs(X_fft(1:N/2)); % fft normalized mag

X_fft_zp = fft(x,N2); % w/ zero padding fft
X_mag_zp = (2/N)*abs(X_fft_zp(1:N2/2)); % w/  zero
    padding fft mag

subplot(2,1,1);
plot(f(1:N/2), X_mag, 'b', 'linewidth', 2, 'DisplayName
    ', 'No padding'); % fft mag
set(gca, 'xlim', [0 0.5], 'ylim', [0 0.06]);
xlabel('Frequency (Hz)');
ylabel('Amplitude');
legend('Orientation','vertical','Box','on','Location','
    southoutside');
grid on;

subplot(2,1,2);
plot(f2(1:N2/2), X_mag_zp, 'k', 'linewidth', 2, '
    DisplayName', '256 padding'); % w/ zero padding fft
    mag
set(gca, 'xlim', [0 0.5], 'ylim', [0 0.06]);
xlabel('Frequency (Hz)');
ylabel('Amplitude');
legend('Orientation','vertical','Box','on','Location','
    southoutside');
grid on;

sgtitle('Zero Padding Effect', 'Interpreter', 'latex');
saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

According to the Figure 5, it's possible to note the interpolation effect provided by zero padding. The blue curve, that is a very short signal with 32 samples, is more abrupt in your way because the MATLAB provides a straight line between the points of the array. In contrast, when is used a zero padding of 256 samples, the black curve turns smoother as a result of the interpolation. It's important to mention that this zero padding interpolation don't add new information to the signal neither generates a better resolution of the signal, but only gets a better spectrum visualization.

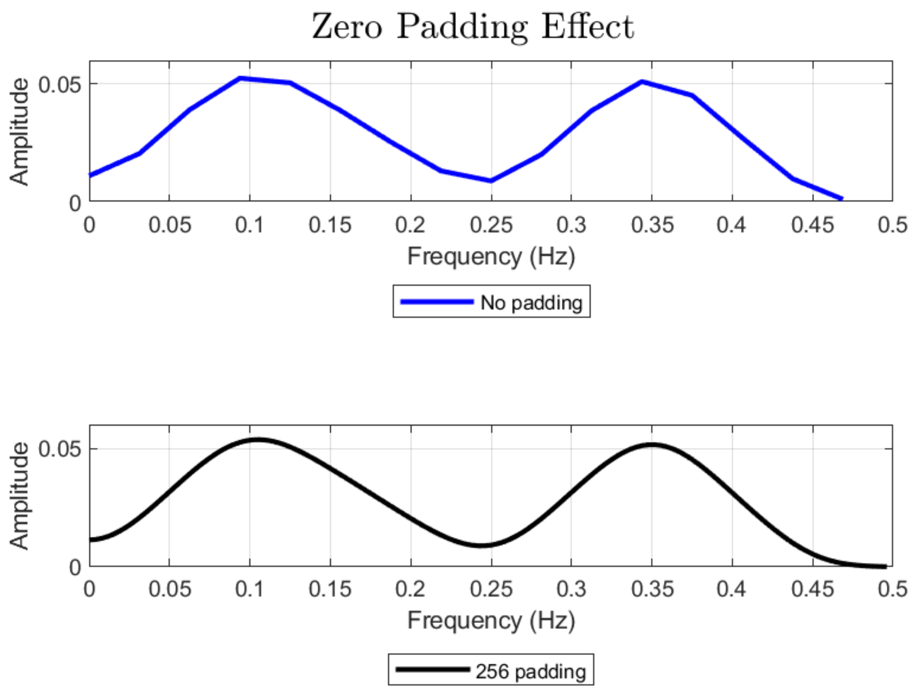

Figure 5: Zero Padding Effect.

1.5 Exercise 3.20:

This and the next problem demonstrate a reduction in energy produced by the use of tapering windows. Use sig-noise to generate a waveform containing two sinusoids, 200 and 300 Hz, with an SNR of -3 dB. Make the waveform fairly short: N = 64 (fs = 1 kHz). Take the magnitude spectrum with a rectangular and a Blackman–Harris window and compare. Plot both magnitude spectra (as always, only valid points). Note the lower values for the windowed data. Rescale the time-domain signal to compensate for the reduction in energy caused by the application of the Blackman–Harris window. Replot both spectra and compare. This problem demonstrates that rescaling the time-domain signal when applying a tapering window is essential if magnitude spectra amplitude levels are to be preserved. Note that the spectrum of the rescaled windowed signal now has approximately the same amplitude as the unwindowed signal. Also note the smoothing done by the Blackman–Harris window. [Hint: Use either Equation 3.27 or MATLAB (w = blackmanharris(N);) to generate the window. Rescale the windowed time signal by multiplying by a scale factor. The scale factor can be determined by taking the area (i.e., sum) of a rectangular window of amplitude 1.0 divided by the area (sum) under the Blackman–Harris window. The rectangular window can be created using MATLAB's ones function.]

The MATLAB's code:

```
close all
fs = 1000; % Sampling frequency
N = 64; % Number of samples
[x, t] = sig_noise([200 300],-3,N); % Noisy signal
X1 = fft(x); % FFT of unwindowed signal
X2 = fft(x.*blackmanharris(N)'); % FFT of signal with
    blackmanharris window
X2_mag = (2/N)*abs(X2(1:N/2));  % Magnitude spectrum of
    windowed signal
X1_mag = (2/N)*abs(X1(1:N/2));  % Magnitude spectrum of
    unwindowed signal
f = (0:N/2-1)*fs/N; % Frequncy vector

subplot(2,1,1)
plot(f,X2_mag); hold on
plot(f,X1_mag);
legend('Blackman-Harris','Rectangular');
ylabel('Magnitude Spectrum')
xlabel('Frequency (Hz)');
title('Before rescaling');

scale_factor = N/sum(blackmanharris(N));  % scale
    factor
X3 = fft(scale_factor*x.*blackmanharris(N)'); % FFT of
```

```
    the scaled windowed signal
X3_mag = (2/N)*abs(X3(1:N/2)); % FFT of the scaled
    signal with blackmanharris window applied

subplot(2,1,2)
plot(f,X3_mag); hold on
plot(f,X1_mag);
legend('Blackman-Harris','Rectangular');
ylabel('Magnitude Spectrum')
xlabel('Frequency (Hz)');
title('After rescaling');

saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

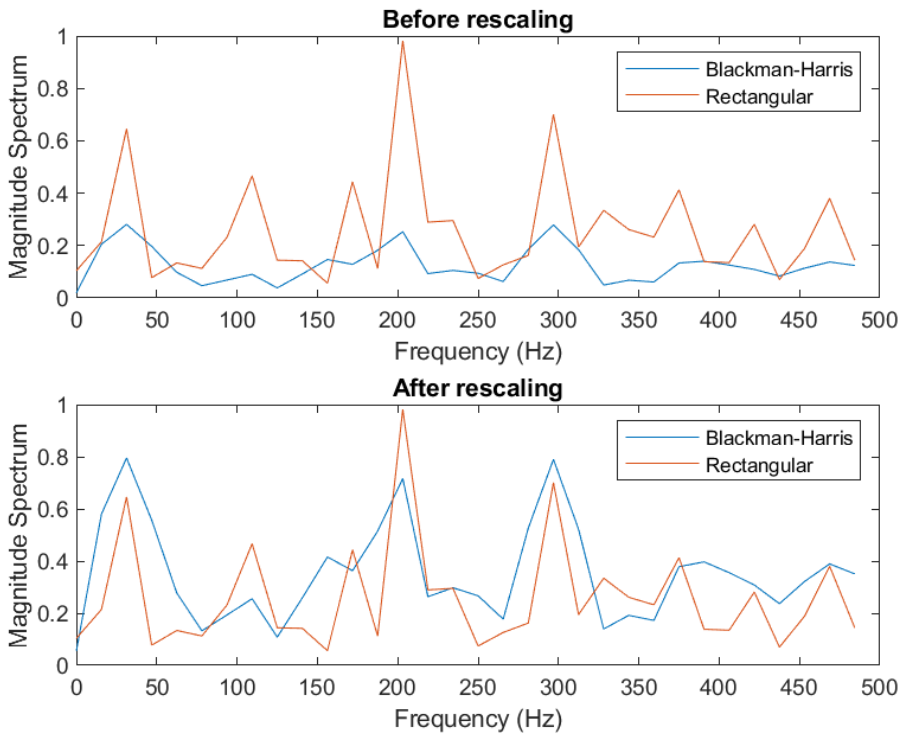

Figure 6: The Rescaling Effect.

It is noticeable that the rescaling of the Blackman-Harris window had a great effect on the signal to preserve the amplitude values.

1.6 Exercise 3.23:

Generate the “filtered” waveform used in Problem 2.36. First, construct a 512-point Gaussian noise array, then filter it by averaging segments of three consecutive samples to produce a new signal. In other words, construct a new vector in which every point is the average of the preceding three points in the noise array: y[n] = x[n]/3 + x[n–1]/3 + x[n–2]/3. This is called a moving average (MA) filter. You could write the code to do this, but an easier way is to convolve the original data with a function consisting of three equal coefficients having a value of 1/3, in MATLAB: h(n) = [1/3 1/3 1/3]. Find the PS of this filtered noise data using the direct method (Equation 3.34). Assume fs = 200 Hz for plotting. Note the lowpass characteristic of this averaging process, as some of the higher frequencies have been removed from the white noise.

The MATLAB's code:

```
clc; clear all; close all;
%% creating array
N = 512; % Number of data points
rng('default'); %defines random number generator to
% Mersenne Twister generator with seed 0
s = rng; %saves to variable s
x= randn(1,N); %creates normal distribution array with
    N points

%% setting
fs = 200; % Sample frequency (200 Hz)
f = (0:N-1)*fs/N; %frequency vector

%% filtering
%filtered signal
h_n = [1/3,1/3,1/3]; %filter that performs moving
    average of every 3 points
x_filtered = conv(x,h_n); %convolution with moving
    average filter

%% magnitude
X_mag = 2*abs(fft(x))/N; %magnitude
X_mag_filtered = 2*abs(fft(x_filtered))/N; %magnitude
```

```
filtered

%% power spectrum
PS_unfiltered = abs(fft(x).^2)/length(x); %power spectrum as eq. 3.34
PS_filtered = abs(fft(x_filtered).^2)/length(x_filtered); %power spectrum as eq. 3.34

%% plot
sgtitle('Unfiltered and filtered magnitude and power spectrum')
subplot(2,1,1)
hold on;
grid();
plot(f(1:N/2),X_mag(1:N/2),'DisplayName','Unfiltered signal');
plot(f(1:N/2),X_mag_filtered(1:N/2),'DisplayName','Filtered signal');
legend('Orientation','horizontal','Box','on','Location','southoutside')
ylabel('Magnitude Spectrum')
xlabel('Frequency (Hz)');

subplot(2,1,2)
hold on;
plot(f(1:N/2),PS_unfiltered(1:N/2),'DisplayName','Unfiltered signal');
plot(f(1:N/2),PS_filtered(1:N/2),'DisplayName','Filtered signal');
grid();
legend('Orientation','horizontal','Box','on','Location','southoutside');
ylabel('Power Spectrum');
xlabel('Frequency (Hz)');

saveas(gcf,'ex3_23.png')
```

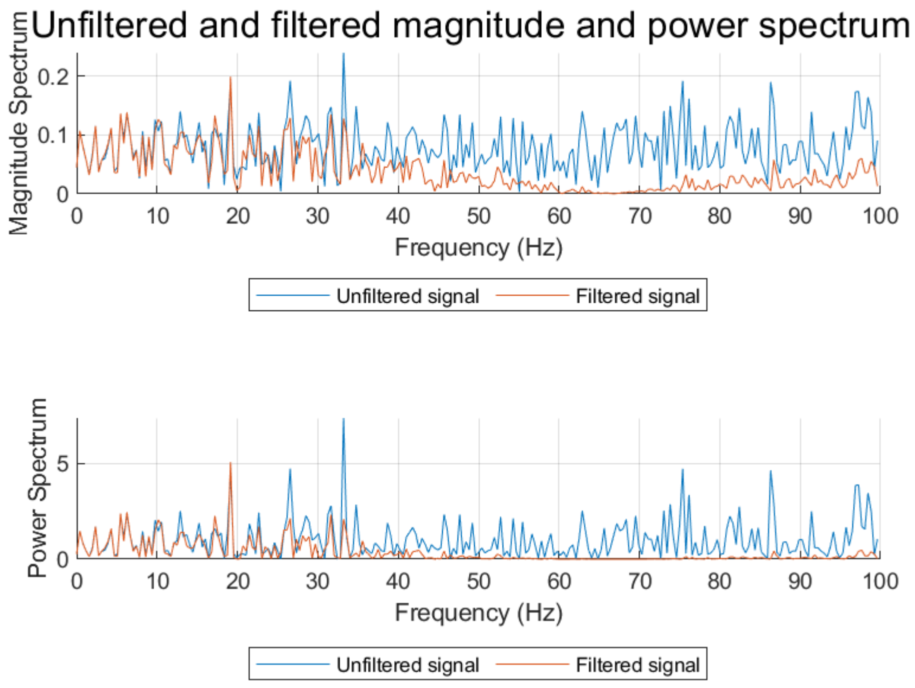

Figure 7: Unfiltered and filtered magnitude and power spectrum of normal distributed array

1.7 Exercise 3.24:

Repeat Problem 3.23 above using the Welch averaging method with 256 and 32 segment lengths. Use the default window and maximum overlap. Use subplot to plot the two power spectra on the same page for comparison. The actual frequency characteristics of the MA filter begin to emerge when the short window is used.

The MATLAB's code:

```
clc; close all; clear all;

N = 512; % number of points - gaussian noise array
 fs = 200; % sampling frequency
gn = randn(1,N); % gaussian noise array

```

```
N1 = 256; % Welch length 1
N2 = 32; % Welch length 2

% PS(f) = |X(f)|^2
[gn_fft1, f1] = pwelch(gn, N1, N1-1, [], fs); %
    periodogram1
[gn_fft2, f2] = pwelch(gn, N2, N2-1, [], fs); %
    periodogram2

subplot(2,1,1);
plot(f1, gn_fft1, 'linewidth', 1); % plot periodogram1
xlabel('Frequency (Hz)');
ylabel('$|P_{s}(f)|$', 'Interpreter', 'latex', '
    Fontsize', 12);
title('\textbf{Periodogram N = 256}', 'Interpreter', '
    latex', 'Fontsize', 12);
grid on;

subplot(2,1,2);
plot(f2, gn_fft2, 'linewidth', 1); % plot periodogram2
xlabel('Frequency (Hz)');
ylabel('$|P_{s}(f)|$', 'Interpreter', 'latex', '
    Fontsize', 12);
title('\textbf{Periodogram N = 32}', 'Interpreter', '
    latex', 'Fontsize', 12);
grid on;

saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

According to the Figure 8, it's possible to visualize the use of the Welch Averaging Method in a Gaussian Noise Signal with $N = 256$ and $N = 32$ points respectively to represent their power spectra. This method is an averaged technique which generates a periodogram. The figure shows that, as bigger as N is, less smooth the shape of the curve is, and it happens because N is the represented window of the signal in the Welch Averaging.

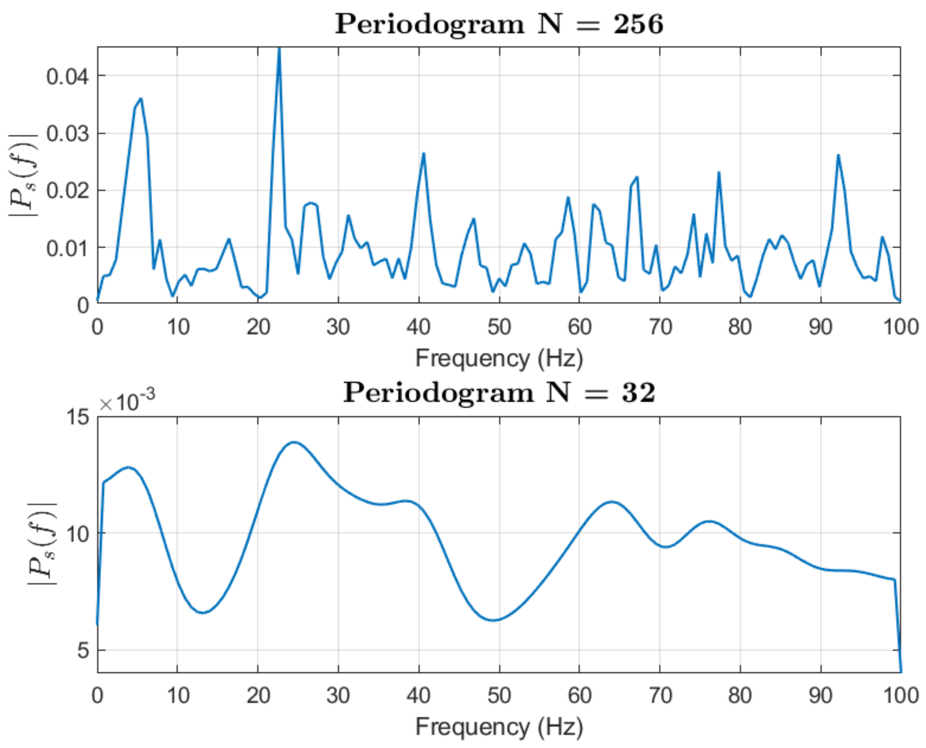

Figure 8: Periodogram of Welch Averaging Method of 256 and 32 points.

1.8 Exercise 3.28:

Use sig-noise to generate a signal that contains a combination of 200- and 300- Hz sine waves with SNRs of -4 dB. Make the data segment fairly short, 64 samples, and recall fs = 1 kHz. Plot the magnitude spectrum obtained with a rectangular, a Hamming, and a Blackman–Harris window. Generate 512-sample Hamming and Blackman–Harris windows using Equations 3.26 and 3.27. Multiply these windows point by point with the signal. Plot the magnitude spectra of the three signals. Note that the difference is slight, but could be significant in certain situations.

The MATLAB's code:

```
% exercise 3.28 - Semmlow
clc; close all; clear all;

fs = 1e3; % sampling frequency
```

```
samples = [64,512]; % array of samples
windows = 'Rectangular  Blackman Hamming';
windows = string(split(windows)'); % windowed filter
    names
pos = 1; % counter used in for loop to plot

for N = samples
    [s, t] = sig_noise([200,300], -4, N); % signal with
        noise
    filters = [ones(1,N); blackman(N)'; hamming(N)']; %
        filters functions
    f = (0:N-1)*(fs/N); % frequency axis

    for i = 1:3
        subplot(2, 3, pos);
        sf = s.*filters(i,:); % appling filter
        S_mag = fft(sf); % fft
        S_mag = (2/N)*abs(S_mag(1:N/2)); % fft
            normalized mag
        plot(f(1:N/2), S_mag, 'linewidth', 1.1); % only
            positive freqs plot
        xlabel('Frequency (Hz)');
        ylabel('Magnitude Spectrum');
        grid on;
        title(sprintf('%s | N = %d', windows(i), N));
        pos = pos + 1; % ++counter to subplot
    end
end

sgtitle('Comparison of windowed filters w/ N = [64, 512
    ]', 'Interpreter', 'latex');
saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

According to the Figure 9, it's notorious the difference between 64 and 512 samples in the three types of windows applied to the signal. The 512-samples has a better resolution of the signal spectrum and of the existing frequencies than the 64-samples, this better resolution happens because de sampling frequency is constant and only N changes (Equation 2) and generates a smaller step, f_i, between the N points. The difference of the plotted magnitude spectra of the three windowed filters is slightly, there's only a quite difference between the amplitude value and some points, but certainly there's a difference. The windowed filters (Rectangular, Hamming and Blackman) have their own particularities and the most appropriate varies according to the situation.

$$f_{axis} = (0 : N - 1) \cdot f_i, \quad f_i = \frac{f_s}{N} \tag{2}$$

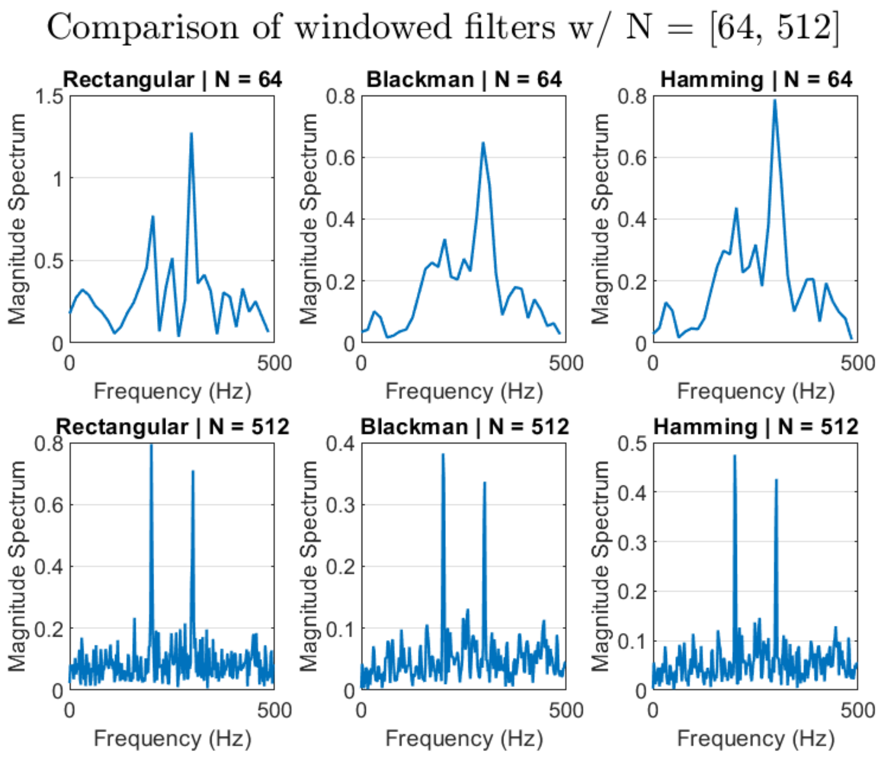

Figure 9: Comparison of windowed filters with different values of N

1.9 Exercise 3.29:

Use sig-noise to generate a signal that contains a combination of 280- and 300-Hz sine waves with SNRs of -10 dB. Make N = 512 samples and recall fs = 1 kHz. Repeat Problem 3.19 and plot the magnitude spectrum obtained with a rectangular, a Hamming, and a Blackman–Harris window. Generate a 512-sample Hamming and Blackman–Harris windows using Equations 3.26 and 3.27. Multiply point by point with the signal. Plot the magnitude spectra of the three signals. Rerun the program several times and observe the variation in spectra that are produced due to variations in the random noise.

Note how the Hamming window is a good compromise between the rectangular window (essentially no window) and the Blackman–Harris windows, again motivating its use as the default window in some MATLAB routines.

The MATLAB's code:

```
close all; clear variables; clc;
fs = 1000; % Sampling frequency
N = 512; %number of poimts
SNR = -10; %SNR (dB)
noise_frequencies = [280 300];
f = (0:N-1)*fs/N; % frequency vector



figure;
tlt = tiledlayout(3, 3);
for j=1:3

[x,t] = sig_noise(noise_frequencies,SNR,N); % Generate
    noise array
X_mag = 2*abs(fft(x))/N; % Magnitude spectrum:
    Retangular Window

% Plot
nexttile
plot(f(1:N/2),X_mag(1:N/2));
ylabel('Magnitude Spectrum');
xlabel("Frequency (Hz)");
grid();
title('Retangular Window');
%

x1 = x .*hamming(N)'; % Janela de Hamming(Eq. 3.26)
X_mag = 2*abs(fft(x1))/N; % Magnitude spectrum: Hamming
    Window

% Plot
nexttile
plot(f(1:N/2),X_mag(1:N/2));
```

```
ylabel('Magnitude Spectrum');
xlabel("Frequency (Hz)");
grid();
title('Hamming Window');

%
x1 = x .*blackmanharris(N)'; % Blackman-Harris Window (
    Eq. 3.27)
X_mag = 2*abs(fft(x1))/N; % Magnitude spectrum:
    Blackman-Harris Window

% Plot
nexttile
plot(f(1:N/2),X_mag(1:N/2));
ylabel('Magnitude Spectrum');
xlabel("Frequency (Hz)");
grid();
title('Blackman-Harris Window');
end

saveas(gcf,'ex3_29.png');
```

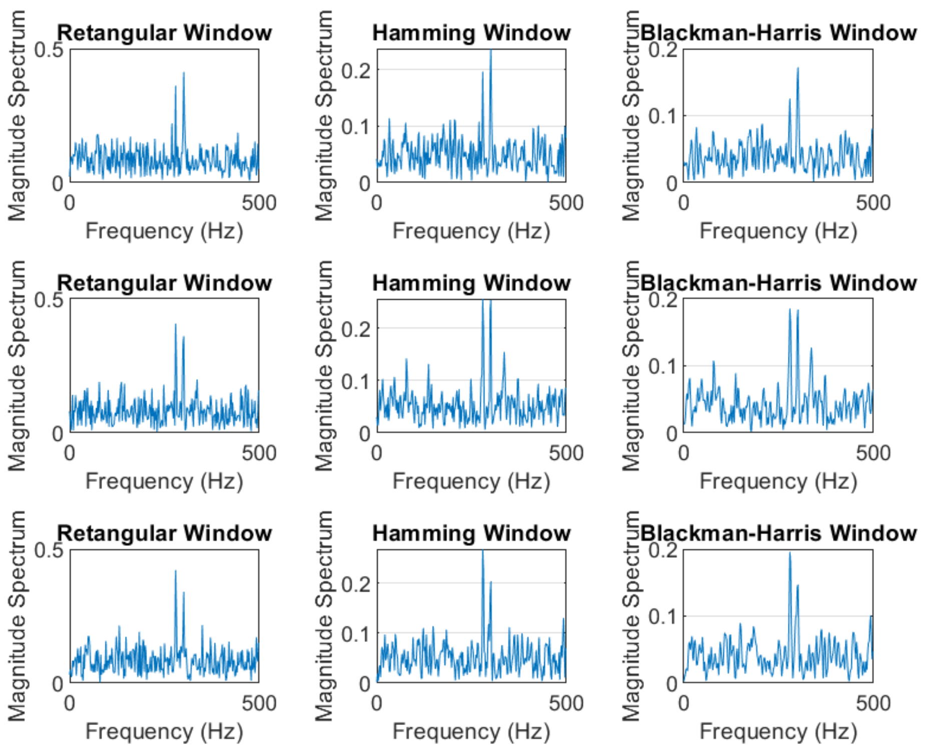

Figure 10: Noise with rectangular, Hamming and Blackman-Harris windows

Every line of Figure 10 represents one iteration of random noise generation windowed with every filter, in order to compare the variation in spectra .

1.10 Exercise 3.33:

Load the file eeg-data.mat that contains an EEG signal in variable eeg (fs = 50 Hz). Analyze these data using the unaveraged power spectral technique and an averaging technique using the pwelch routine. Find a segment length that gives a smooth background spectrum, but still retains any important spectral peaks. Use a 99% overlap.

The MATLAB's code:

```
clc; close all; clear all;
warning('off');
load eeg_data.mat % variable eeg

```

```
fs = 50; % sampling frequency
N = length(eeg); % length of egg signal
f = (0:N-1)*(fs/N); % frequency axis

% empiric method to see a smooth spectrum with
    information of the peaks
Ns = N/14; % segment element ~ 58
unavg_ps = abs( fft(eeg).^2 )/N; % unaveraged technique
[avg_ps, f1] = pwelch(eeg, Ns, floor(0.99*Ns), [], fs);
     % averaged technique

subplot(2,1,1);
plot(f(1:N/2), unavg_ps(1:N/2)); % only positive freqs
    (N/2)
xlabel('Frequency (Hz)');
ylabel('Power Spectrum');
title('\textbf{Standard Spectrum}', 'Interpreter', '
    latex', 'Fontsize', 12);
set(gca, 'xlim', [0 25], 'ylim', [0 12e7]);
grid on;

subplot(2,1,2);
plot(f1, avg_ps);
xlabel('Frequency (Hz)');
ylabel('Power Spectrum');
title('\textbf{Periodogram}', 'Interpreter', 'latex', '
    Fontsize', 12);
set(gca, 'xlim', [0 25], 'ylim', [0 12e5]);
grid on;

saveas(gcf, sprintf('%s.png', mfilename)); % save image
```

In the figure 11, two different aproacches of power spectrum analysis are plotted: the unaveraged technique and the averaged thechnique. The unaveraged technique represents a standard power spectrum of the EEG signal without averaging, this approach shows more precisely the amplitude of the existing frequencies of the biosignal, but also shows a lot of noise. On the other and the averaged technique represents the power spectrum with averaging (called of periodogram), and it reflects more clearly the constant energy of the white noise, but it's notorious the loss of frequency information. Thus, the most suitable method depends on situation and what information is required to exctract.

The periodogram represents the power spectrum with the averaging of N points, and this number of points defined. In this case, the number of points that gives a smooth background spectrum, but still retains any important peaks was called of $N_s = \dfrac{N}{14} \approx 58$, considering $N = 801$ as the length of EEG signal, as written in the code above.

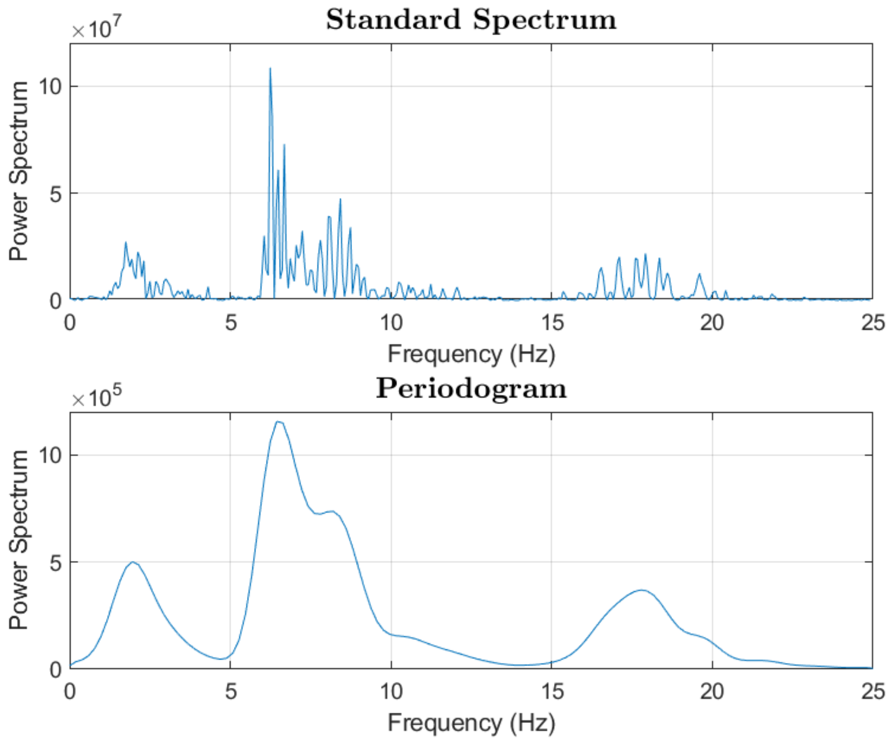

Figure 11: Standard Spectrum and Periodogram of an EEG signal.