Voice Signal Filtering with IIR and FIR Filters

Sofia Noemi Crobeddu
student number: 032410554
International Master of Biometrics and Intelligent Vision - Université Paris-Est Cretéil (UPEC)
Assignment 5

Introduction

In this project we analyze the differences between IIR and FIR filters, varying their order and applying them to clean and noisy signal. Metrics and parameters in order to give a suitable interpretation are performed.

1 Uploading and Processing of a Voice Signal

For this analysis we can upload the speech signal recorded Assignment 2 (*Speech Processing in Frequency domain Using Periodogram*). Through the function *wavfile* of the library *Scipy*, we can read the voice signal and take also the sampling frequency (fs). It is important also to specify that fs was set at priori, before the recording, to 44.1 kHz, a common measure to perform good analysis.
Below it is showed the code performed for this starting step:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.io.wavfile as wav

#We can take the speech recorded for Assignment 2 (option 2)
fs, speech_signal = wav.read('C:/Users/sofyc/OneDrive/Desktop/UPEC/Data capture and processing/
    assignment 2/recording_my_speech.wav')

#Checking of sampling frequency (fs) that was set to sampling frequency = 44.1 kHz
print(f'{fs} Hz')
#44100 Hz = 44.1 kHz
```

After reading data, we can find the duration of the signal. Again, as for fs, it was set a priori to 10 seconds. Anyway, the code below shows the commands to calculate the duration, expressed as:

$$Duration = \frac{\text{Number of samples}}{fs}$$

```
#Duration = (number of samples) / (sampling frequency)
duration = len(speech_signal)/fs
print(f'{duration} seconds')
```

Figure 1 shows the original speech signal of 10 seconds.

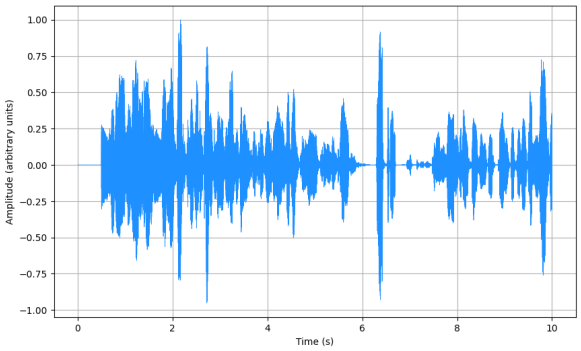

Fig. 1: Speech Signal.

After this preliminary analysis, we can go on calculating the Fast Fourier Transform (FFT), an efficient algorithm to compute the Discrete Fourier Transform (DFT). The DFT is defined as

$$X[k] = (w_k, x) = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \ , \ k = 0, ..., N-1$$

where w_k are the Fourier basis, and x is the speech signal.
The FFT converts the signal in time domain to a signal in frequency domain, giving the spectrum as the output. We can perform this through the function *fft* from *Scipy* library. We also take the frequencies associated to the samples of the spectrum though the function *fftfreq*. In this case, since we are dealing with a real signal (composed by R values), the signal will be symmetric in respect to 0. For this reason we save just the frequencies in the positive halp of the spectrum.
After calculating the frequencies of the spectrum, we calculate the magnitude, in order to have a final plot with the spectrum representation. The magnitude is expressed as $|X[k]|^2$ and it is a measure of the energy of the signal. The Figure 2 shows the frequency spectrum using this magnitude just calculated, that represents the amplitude of the spectrum.
In order to have a better visualization, we can convert the magnitude in decibel, since dB highlights the amplitudes' variations in the spectrum. The final graph is shown in Figure 3, that represents the same frequency spectrum graph but with a different measure unit in the y axis (magnitude axis).
The code to perform this analysis is shown below:

```
from scipy.fft import fft, fftfreq

#Applying the FFT
spectrum = fft(speech_signal) #Spectrum of the signal

#We take the correspondent frequencies of the samples of the Fourier transform
freq_FFT = fftfreq(len(speech_signal), 1/fs)
#We take just the half positive part of the spectrum's frequencies since they are simmetric to 0
    for real signals
freq_FFT = freq_FFT[: len(speech_signal)//2]

#Magnitude
magnitude = np.abs(spectrum[: len(speech_signal) // 2])
#Transforming the magnitude in decibel
magnitude_dB = 20*np.log10(magnitude)
```

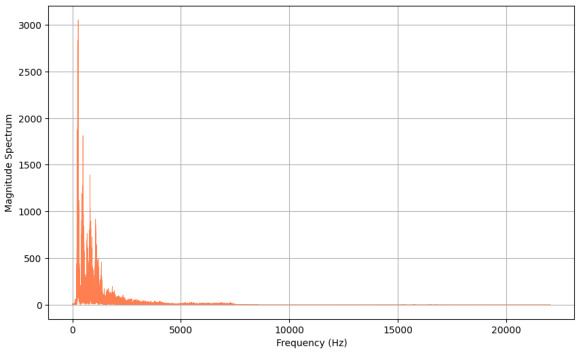

Fig. 2: Frequency Spectrum of the speech signal.

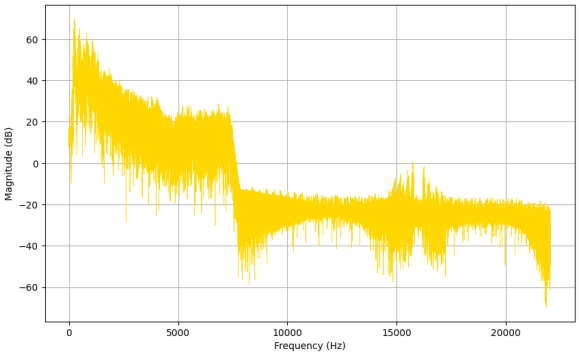

Fig. 3: Frequency Spectrum of the speech signal (dB).

From the graphs of the spectrum we can say that the fundamental frequency, as we could expected, is below 1000 Hz. Moreover, the signal doesn't seem to have much noise and it looks clean.The formants, i.e. the vocal resonances that give to my voice its characteristic timbre, are instead between 1000 and 4000 Hz.

2 Infinite Impulse Response (IIR) Filter Design and Application

In this section, we use the same filtering function applied also in Assignment 1, with the difference that in this case, we will have three types of filters: Low-pass, High-pass and Band-pass. To completeness of information, for the analysis we have to remember that the Low-pass filter put to zero high frequencies and maintains low ones. For the High-pass filter is the contrary, instead for the Band-pass filter only the frequencies inside the band are maintained.
To implement this, first we create three Butterworth filter functions. they are a digital filter that doesn't introduce variation in the signal of the general band-pass, i.e. the range of frequencies we want to maintain. Basically, out of the range, the filter reduces gradually the amplitude of the frequencies. This "lower" answer of the filter doesn't introduce distortions in the filtering of the signal. This implementation is performed through the function *butter* from *Scipy*, which calculates the coefficients of Butterworth filter that will be used to filter the signal.
The cutoff frequency for the Low-pass filter is set to 3000 Hz, while the cutoff frequency for the High-pass filter is set to 500 Hz. The Band-pass filter will be ranged by these two.
Inside the Butterworth filter functions, the cutoff frequency is normalized using the Nyquist theorem (i.e. we divided the cutoff by half of the sampling frequency). This process is perform for each type of filter we are considering, and the code is shown below:

```python
from scipy.signal import butter, lfilter, freqz

#fs = 44100 Hz (sampling frequency)
high_cutoff = 3000  #cutoff frequency (Hz) of Low-pass filter
low_cutoff = 500  #cutoff frequency (Hz) of High-pass filter

#Low-pass Butterworth filter
def lowpass_butterworth(highcut, fs, order):
    nyquist = 0.5 * fs #Nyquist Theorem: f_sampling >= 2*f_max --> f_sampling = fs
    high = highcut / nyquist #normalized frequency
    b, a = butter(order, high, btype='low') #coefficients of Butterworth filter
    return b, a

#High-pass Butterworth filter
def highpass_butterworth(lowcut, fs, order):
    nyquist = 0.5 * fs #Nyquist Theorem: f_sampling >= 2*f_max --> f_sampling = fs
    low = lowcut / nyquist #normalized frequency
    b, a = butter(order, low, btype='high')
    return b, a

#Band-pass Butterworth filter
def bandpass_butterworth(lowcut, highcut, fs, order):
    #higher order means a narrower transition between the maintained (i.e. passed) band, and the chnged band, but this can cause distortions
    #analog=False since it is a digital filter
    nyquist = 0.5 * fs #Nyquist Theorem: f_sampling >= 2*f_max --> f_sampling = fs
    low = lowcut / nyquist #normalized frequency
    high = highcut / nyquist #normalized frequency
    b, a = butter(order, [low, high], btype='band') #coefficients of Butterworth filter
    return b, a
```

In order to analyze the effect of the order of the filter, we need to represent graphically the Frequency Response for each type of filter varying the order. After calculating the coefficients through the Butterworth functions, we use the function *freqz* to obtain the normalized frequencies in rad/seconds, and the magnitude values associated. Since the frequencies are in rad/s, we convert them into Heartz (Hz). The specification *worN=8000* in the *freqz* function is to define the number of frequency points over which the response is calculate, but it is arbitrary. This function calculates the response within 8000 points between 0 and Nyquist frequency.
The code to implement this is shown below:

```python
#Visualization of frequency response of the IIR filters: we can use the butterworth functions.
order_values = [2,4,6]

for filter_type in ['low', 'high', 'band']:
    plt.figure(figsize=(10, 6))
    for order in order_values:
        if filter_type == 'low':
            b, a = lowpass_butterworth(high_cutoff, fs, order)
        elif filter_type == 'high':
            b, a = highpass_butterworth(low_cutoff, fs, order)
        elif filter_type == 'band':
            b, a = bandpass_butterworth(low_cutoff, high_cutoff, fs, order)

        #Frequency response based on filter type
        w, h = freqz(b, a, worN=8000)
        #w: array of normalized frequencies in rad/s, h: magnitude values till those frequencies.

        plt.plot((fs * 0.5 / np.pi) * w, abs(h), label=f"Order {order}")
        #w is converted into Heartz from radiants, while abs(h) takes just the aplitude part of the magnitude.

    plt.title(f"Frequency Response of IIR {filter_type.capitalize()}-pass Filter")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.grid(True)
    plt.legend()
    plt.show()
```

Figures 4, 5, 6 show the Frequency response for each type of filter. In each graph we can see three different lines, one per order (2, 4, 6). As we can see the increase of the order in the filters corresponds to a more evident shift between the Bandpass and the Stopping band. This means that with a higher order, we can mitigate easily the noise, or undesirable frequencies, as in this case. We can also say that the increase of the order doesn't modify the shape of the Frequency Response, however we can see a better selection in terms of frequencies.

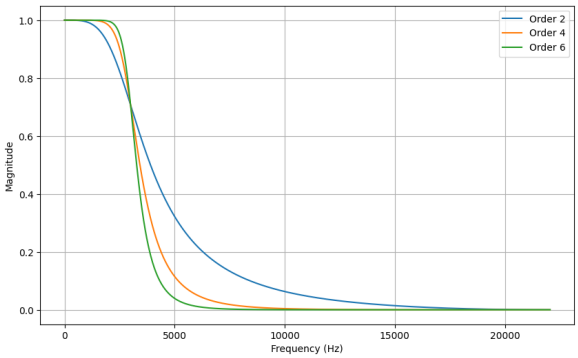

Fig. 4: Frequency Response of IIR Low-pass Filter.

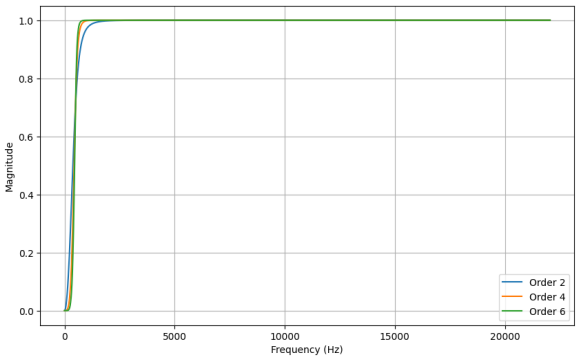

Fig. 5: Frequency Response of IIR High-pass Filter.

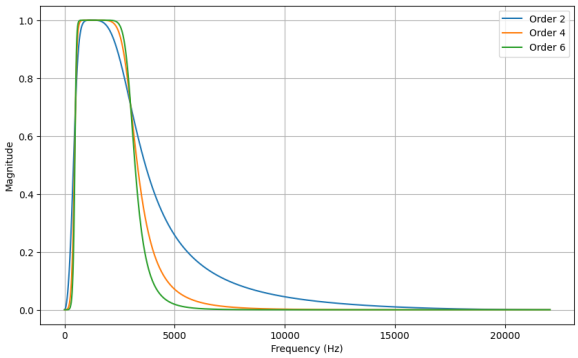

Fig. 6: Frequency Response of IIR Band-pass Filter.

Now, we can define the final IIR filtering functions. After finding the coefficients thorugh the Butterworth functions, we use *lfilter* to apply the filter to the speech signal.
The code to implement them is shown below:

```
#Functions of low-pass, high-pass, band-pass filter: here we use lfilter to apply the filters to
    the signal.

#IIR Low-pass filter
def IIR_lowpass_filter(data, highcut, fs, order):
    b, a = lowpass_butterworth(highcut, fs, order=order)
    y = lfilter(b, a, data)
```

```
    return y

#IIR High-pass filter
def IIR_highpass_filter(data, lowcut, fs, order):
    b, a = highpass_butterworth(lowcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

#IIR Band-pass filter
def IIR_bandpass_filter(data, lowcut, highcut, fs, order):
    b, a = bandpass_butterworth(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y
```

Since we have defining the functions, we can now apply them to the signal.
Below it is shown the code for the application of Low-pass filter for each order. After apply the IIR filter, we also calculate the duration, we apply the FFT and we take the frequencies of the FFT in order to obtain the magnitude. These values are necessary for the plots (code written at the end). We basically repeat the same analysis we did for the original speech signal. The plots of the filtered signal and of the frequency spectrum (both with the magnitude amplitude and the magnitude in dB), are then performed for each case.

```
#Application of the IIR filters:
#IIR Low-pass filtered speech signal
low_magnitude_avg = []

for order in order_values:
    low_filtered_speech = IIR_lowpass_filter(speech_signal, high_cutoff, fs, order)

    #Duration
    duration_low_speech = len(low_filtered_speech) / fs

    #Spectrum
    spectrum_low = fft(low_filtered_speech)

    #Frequencies of the FFT
    freq_FFT_low = fftfreq(len(low_filtered_speech), 1/fs)
    freq_FFT_low = freq_FFT_low[: len(low_filtered_speech)//2]

    #Magnitude
    magnitude_low = np.abs(spectrum_low[: len(low_filtered_speech) // 2])
    #Average magnitude
    l_m_avg = np.mean(magnitude_low)
    low_magnitude_avg.append(l_m_avg)

    #Transforming the magnitude in decibel
    magnitude_low_dB = 20*np.log10(magnitude_low)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_low_speech, len(low_filtered_speech)), low_filtered_speech
    , color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"IIR Low-pass filtered Speech signal (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_low, magnitude_low, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_low, magnitude_low_dB, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the IIR Low-pass filtered signal (Order {order})"
    , ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 7, 8, 9 show the resulting plot from the application of the IIR Low-pass filter to the signal. As we can see, the more is the order, the more is the attenuation of high frequencies (as we expect since we are applying a Low-pass filter). The band in fact becomes shorter. Moreover, with the increase of the order, there is a reduction in the oscillation. This result was also caught by the Frequency Response graph.

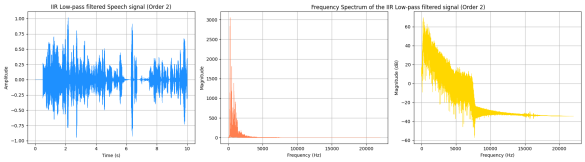

Fig. 7: Plots of IIR Low-pass filtered signal, Order=2.

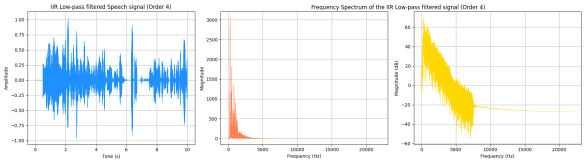

Fig. 8: Plots of IIR Low-pass filtered signal, Order=4.

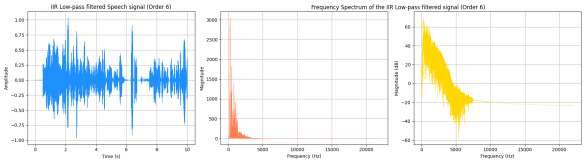

Fig. 9: Plots of IIR Low-pass filtered signal, Order=6.

Our second implementation regards the IIR High-pass filter. The process is the same as before for the IIR Low-pass filter. The code is shown below:

```
#IIR High-pass filtered speech signal
high_magnitude_avg = []

for order in order_values:
    high_filtered_speech = IIR_highpass_filter(speech_signal, low_cutoff, fs, order)

    #Duration
    duration_high_speech = len(high_filtered_speech) / fs

    #Spectrum
    spectrum_high = fft(high_filtered_speech)

    #Frequencies of the FFT
    freq_FFT_high = fftfreq(len(high_filtered_speech), 1/fs)
    freq_FFT_high = freq_FFT_high[: len(high_filtered_speech)//2]

    #Magnitude
    magnitude_high = np.abs(spectrum_high[: len(high_filtered_speech) // 2])
    #Average magnitude
    h_m_avg = np.mean(magnitude_high)
    high_magnitude_avg.append(h_m_avg)

    #Transforming the magnitude in decibel
    magnitude_high_dB = 20*np.log10(magnitude_high)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))
```

```
28
29      #Plot time domain
30      axs[0].plot(np.linspace(0, duration_high_speech, len(high_filtered_speech)),
         high_filtered_speech, color='dodgerblue', linewidth=0.5)
31      axs[0].set_title(f"High-pass filtered Speech signal (Order {order})")
32      axs[0].set_xlabel("Time (s)")
33      axs[0].set_ylabel("Amplitude")
34      axs[0].grid(True)
35
36      #Plot Magnitude Spectrum
37      axs[1].plot(freq_FFT_high, magnitude_high, color="coral", linewidth=0.5)
38      axs[1].set_xlabel("Frequency (Hz)")
39      axs[1].set_ylabel("Magnitude")
40      axs[1].grid(True)
41
42      #Plot Magnitude in dB
43      axs[2].plot(freq_FFT_high, magnitude_high_dB, color="gold", linewidth=0.5)
44      axs[2].set_xlabel("Frequency (Hz)")
45      axs[2].set_ylabel("Magnitude (dB)")
46      axs[2].grid(True)
47
48      #Title of the second and third graphs
49      fig.text(0.68, 0.94, f"Frequency Spectrum of the IIR High-pass filtered signal (Order {order})
         ", ha='center', fontsize=12, color="black")
50
51      #Space between subplots
52      plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
53      plt.tight_layout()
54      plt.show()
```

Figures 10, 11, 12 show the resulting plot from the application of the IIR High-pass filter to the signal. As we can see, the more is the order, the less is the amplitude in low frequencies. As we could expect, the spectrum shows an attenuation of low frequencies. Also, the higher the order, the more the cutoff frequency moves to lower values. As for the previous case, a higher order implies a more evident shift between the pass-band and the stop-band.
We can also notice that the waveform of the filtered signal becomes more similar to a derivative of the original signal the more is the order, since low frequencies corresponds to slow variations of the signals and are gradually removed.

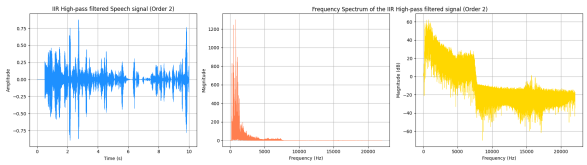

Fig. 10: Plots of IIR High-pass filtered signal, Order=2.

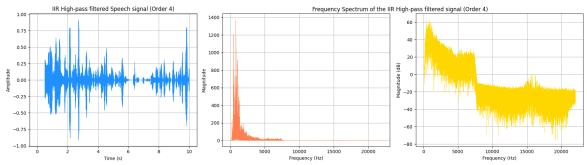

Fig. 11: Plots of IIR High-pass filtered signal, Order=4.

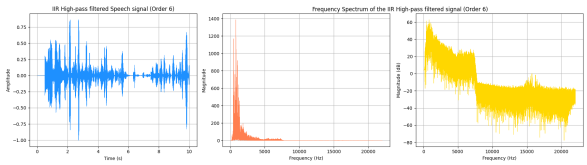

Fig. 12: Plots of IIR High-pass filtered signal, Order=6.

As for the previous case, the implementation of the IIR Band-pass filter is the same. The code is shown below:

```
#IIR Band-pass filtered speech signal
band_magnitude_avg = []

for order in order_values:
    band_filtered_speech = IIR_bandpass_filter(speech_signal, low_cutoff, high_cutoff, fs, order)

    #Duration
    duration_band_speech = len(band_filtered_speech) / fs

    #Spectrum
    spectrum_band = fft(band_filtered_speech)

    #Frequencies of the FFT
    freq_FFT_band = fftfreq(len(band_filtered_speech), 1/fs)
    freq_FFT_band = freq_FFT_band[: len(band_filtered_speech)//2]

    #Magnitude
    magnitude_band = np.abs(spectrum_band[: len(band_filtered_speech) // 2])
    #Average magnitude
    b_m_avg = np.mean(magnitude_band)
    band_magnitude_avg.append(b_m_avg)

    #Transforming the magnitude in decibel
    magnitude_band_dB = 20*np.log10(magnitude_band)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_band_speech, len(band_filtered_speech)),
     band_filtered_speech, color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"Band-pass filtered Speech signal (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_band, magnitude_band, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_band, magnitude_band_dB, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the IIR Band-pass filtered signal (Order {order})
    ", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 13, 14, 15 show the resulting plots from the application of the IIR Band-pass filter to the signal. As we can see, the waveform is more selective and is concentrated on the band-pass. Same for energy, that is concentrated on this range. As for the previous IIR filters, the increase of the order causes a more evident shift between the passband and the stopband. Also, compared to the Low-pass and the High-pass filters, this one seems to isolate better the spectrum, in fact the the harmonics are removed.

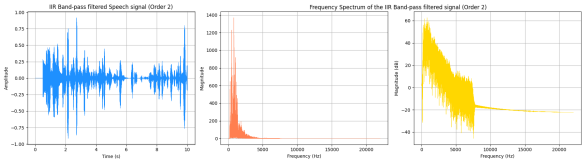

Fig. 13: Plots of IIR Band-pass filtered signal, Order=2.

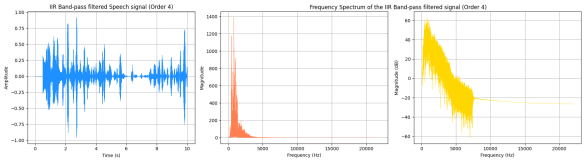

Fig. 14: Plots of IIR Band-pass filtered signal, Order=4.

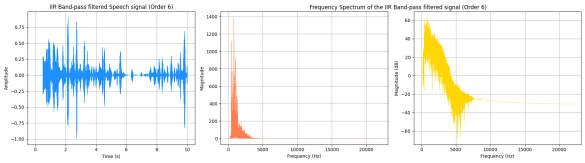

Fig. 15: Plots of IIR Band-pass filtered signal, Order=6.

In order to have a better interpretation of these results, we can analyze the smoothness through the standard deviation of the consecutive differences of the filtered signals. These consecutive differences calculated through the function *diff* of *Numpy*, reflects the instant variations of the amplitude of the signal. The standard deviation of them represents their variability. A low value of smoothness indicates more instant variations and less uniformity. In this sense we can evaluate the effect of the filter looking at the attenuation and also the signal uniformity.
Moreover, it is also calculated the magnitude for each case.
For both smoothness and magnitude, the results are stored in a dictionary and after in a dataframe. The code implemented is shown below:

```
#Other calculations for the final interpretation and analysis
#Smoothness through Standard Variation
smoothness_low = []
smoothness_high = []
smoothness_band = []

for order in order_values:
    low_filtered_speech = IIR_lowpass_filter(speech_signal, high_cutoff, fs, order)
    s_low = np.std(np.diff(low_filtered_speech))
    smoothness_low.append(s_low)

    high_filtered_speech = IIR_highpass_filter(speech_signal, low_cutoff, fs, order)
    s_high = np.std(np.diff(high_filtered_speech))
    smoothness_high.append(s_high)

    band_filtered_speech = IIR_bandpass_filter(speech_signal, low_cutoff, high_cutoff, fs, order)
    s_band = np.std(np.diff(band_filtered_speech))
    smoothness_band.append(s_band)

res_smoothness = {
    "Order": order_values,
    "Low-pass": [round(s, 4) for s in smoothness_low],
    "High-pass": [round(s, 4) for s in smoothness_high],
    "Band-pass": [round(s, 4) for s in smoothness_band]
}

res_smoothness_df = pd.DataFrame(res_smoothness)

#Visualizing the magnitude averages
res_magnitude = {
    "Order": order_values,
    "Low-pass": [round(s, 4) for s in low_magnitude_avg],
    "High-pass": [round(s, 4) for s in high_magnitude_avg],
    "Band-pass": [round(s, 4) for s in band_magnitude_avg]
}

res_magnitude_df = pd.DataFrame(res_magnitude)
```

Tables 1 and 2 show the results. The smoothness table shows a general stability for every filter, but the Low-pass filter has higher smoothness compared to the others. Pass-band filter seems to capture

more variations than the others in terms of smoothness.
Regarding the magnitude we can say that it decrease gradually the higher the order, especially for the Low-pass and Band-pass filters. as we saw from the previous graphs, a higher order produces a better attenuation. Band-pass filter shows a better reduction.

Order	Low-pass	High-pass	Band-pass
2	0.0121	0.0117	0.0112
4	0.0122	0.0119	0.0112
6	0.0122	0.0120	0.0112

Table 1: Values of Smoothness for IIR Low-pass, IIR High-pass, and IIR Band-pass filters at different orders.

Order	Low-pass	High-pass	Band-pass
2	17.4776	14.2266	13.0401
4	17.4439	13.6328	12.1620
6	17.4142	13.3484	11.8064

Table 2: Values of Magnitude for IIR Low-pass, IIR High-pass, and IIR Band-pass filters at different orders.

3 Finite Impulse Response (FIR) Filter Design and Application

In this section we implement instead the FIR filters. FIR filters are still digital filters but they have a finite impulse response. This means that, when applied to an impulse, the filter gives a response that lasts in a finite time range. Compared to IIR filters, FIR ones are more stable and are more predictable.
As for IIR filters, we implement the Low-pass, High-pass and Band-pass filter. This time we take the coefficients through the function *firwin*. The argument *order + 1* determines how much large is the filter, i.e. the number of coefficients. Also, as for the Butterworth function for the IIR filters, we normalize again using the Nyquist theorem. Finally, the signal is filtered as for IIR case, using the function *lfilter*.
The code to implement the functions is shown below:

```
from scipy.signal import firwin

#Creation of the FIR filters
#FIR Low-pass filter
def FIR_lowpass_filter(data, highcut, fs, order):
    coefficients = firwin(order + 1, highcut / (0.5 * fs), pass_zero=True)
    y = lfilter(coefficients, 1.0, data)
    return y

#FIR High-pass filter
def FIR_highpass_filter(data, lowcut, fs, order):
    coefficients = firwin(order + 1, lowcut / (0.5 * fs), pass_zero=True)
    y = lfilter(coefficients, 1.0, data)
    return y

#FIR Band-pass filter
def FIR_bandpass_filter(data, lowcut, highcut, fs, order):
    coefficients = firwin(order + 1, [lowcut / (0.5 * fs), highcut / (0.5 * fs)], pass_zero=False)
    y = lfilter(coefficients, 1.0, data)
    return y
```

Now that we have the functions, we can apply them to the signal. The same process of application used in section 2 for IIR filters, is performed. The only difference is that for this FIR filters we use

different orders: 20, 50, 100.
The code to apply the FIR Low-pass filter is the following one:

```
order_values2 = [20, 50, 100]

#FIR Low-pass filtered speech signal
for order in order_values2:
    low_filtered_FIR = FIR_lowpass_filter(speech_signal, high_cutoff, fs, order)

    #Duration
    duration_low_FIR = len(low_filtered_FIR) / fs

    #Spectrum
    spectrum_low_FIR = fft(low_filtered_FIR)

    #Frequencies of the FFT
    freq_FFT_low_FIR = fftfreq(len(low_filtered_FIR), 1/fs)
    freq_FFT_low_FIR = freq_FFT_low_FIR[: len(low_filtered_FIR)//2]

    #Magnitude
    magnitude_low_FIR = np.abs(spectrum_low_FIR[: len(low_filtered_FIR) // 2])

    #Transforming the magnitude in decibel
    magnitude_low_dB_FIR = 20*np.log10(magnitude_low_FIR)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_low_FIR, len(low_filtered_FIR)), low_filtered_FIR, color='
     dodgerblue', linewidth=0.5)
    axs[0].set_title(f"FIR Low-pass filtered Speech signal (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_low_FIR, magnitude_low_FIR, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_low_FIR, magnitude_low_dB_FIR, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the FIR Low-pass filtered signal (Order {order})"
     , ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 16, 17 and 18 show the resulting plots from the application of the FIR Low-pass filter to the signal. As we can see, the more is the order, the more is the selection. The transition between the passband and the stopband is in fact very narrow, more than for the IIR Low-pass filter. Also, compared to the IIR one, in this case we have a better waveform and less oscillation.

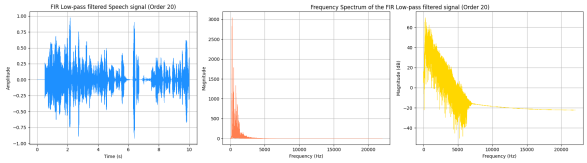

Fig. 16: Plots of FIR Low-pass filtered signal, Order=20.

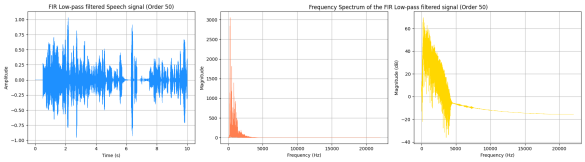

Fig. 17: Plots of FIR Low-pass filtered signal, Order=50.

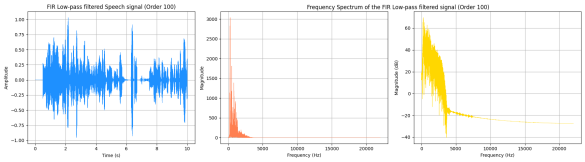

Fig. 18: Plots of FIR Low-pass filtered signal, Order=100.

The same process is applied to the FIR High-pass filter, and the code is shown below:

```
#FIR High-pass filtered speech signal
for order in order_values2:
    high_filtered_FIR = FIR_highpass_filter(speech_signal, low_cutoff, fs, order)

    #Duration
    duration_high_FIR = len(high_filtered_FIR) / fs

    #Spectrum
    spectrum_high_FIR = fft(high_filtered_FIR)

    #Frequencies of the FFT
    freq_FFT_high_FIR = fftfreq(len(high_filtered_FIR), 1/fs)
    freq_FFT_high_FIR = freq_FFT_high_FIR[: len(high_filtered_FIR)//2]

    #Magnitude
    magnitude_high_FIR = np.abs(spectrum_high_FIR[: len(high_filtered_FIR) // 2])

    #Transforming the magnitude in decibel
    magnitude_high_dB_FIR = 20*np.log10(magnitude_high_FIR)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_high_FIR, len(high_filtered_FIR)), high_filtered_FIR,
     color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"FIR High-pass filtered Speech signal (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_high_FIR, magnitude_high_FIR, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_high_FIR, magnitude_high_dB_FIR, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the FIR High-pass filtered signal (Order {order})
    ", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 19, 20 and 21 show the resulting plots from the application of the FIR High-pass filter to the signal. As we can see, with a higher order the filter is more efficient (same as for all the other filters). We have more oscillation in high frequencies, since we are removing low frequencies.

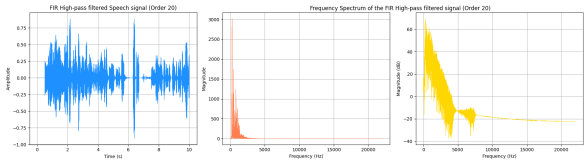

Fig. 19: Plots of FIR High-pass filtered signal, Order=20.

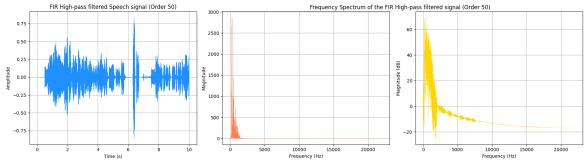

Fig. 20: Plots of FIR High-pass filtered signal, Order=50.

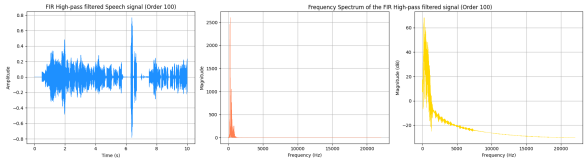

Fig. 21: Plots of FIR High-pass filtered signal, Order=100.

Finally, we repeat the process for the FIR Band-pass filter. The code is shown below:

```
#FIR Band-pass filtered speech signal
for order in order_values2:
    band_filtered_FIR = FIR_bandpass_filter(speech_signal, low_cutoff, high_cutoff, fs, order)

    #Duration
    duration_band_FIR = len(band_filtered_FIR) / fs

    #Spectrum
    spectrum_band_FIR = fft(band_filtered_FIR)

    #Frequencies of the FFT
    freq_FFT_band_FIR = fftfreq(len(band_filtered_FIR), 1/fs)
    freq_FFT_band_FIR = freq_FFT_band_FIR[: len(band_filtered_FIR)//2]

    #Magnitude
    magnitude_band_FIR = np.abs(spectrum_band_FIR[: len(band_filtered_FIR) // 2])

    #Transforming the magnitude in decibel
    magnitude_band_dB_FIR = 20*np.log10(magnitude_band_FIR)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_band_FIR, len(band_filtered_FIR)), band_filtered_FIR,
     color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"FIR Band-pass filtered Speech signal (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
```

```
    axs[1].plot(freq_FFT_band_FIR, magnitude_band_FIR, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_band_FIR, magnitude_band_dB_FIR, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the FIR Band-pass filtered signal (Order {order})
    ", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 22, 23 and 24 show the resulting plots from the application of the FIR Band-pass filter to the signal. As we can see, again we have a better selection and isolation with higher values of the order. In this case we are attenuating both low frequencies and high ones. The evident difference in the graph is in the last one with the Frequency spectrum: we can see how the band is reduced. For this reason, outside the band the magnitude tends to decreases gradually.

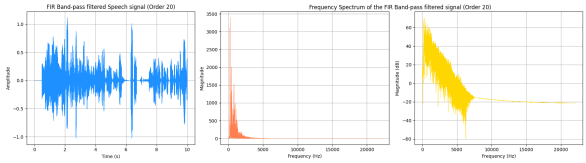

Fig. 22: Plots of FIR Band-pass filtered signal, Order=20.

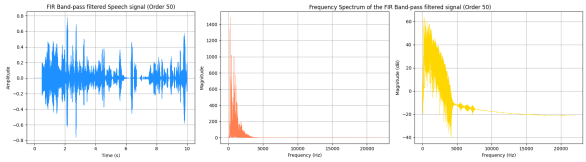

Fig. 23: Plots of FIR Band-pass filtered signal, Order=50.

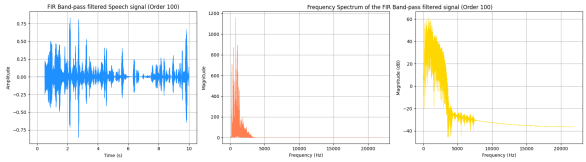

Fig. 24: Plots of FIR Band-pass filtered signal, Order=100.

Now, in order to have a better interpretation of the results, we look at the Frequency response as we did in section 2 for the IIR filters. Hence, we calculate coefficients for every type and every order, and we catch the array of normalized frequencies (w), and the magnitude values (h). After this, we convert w from radiant/seconds to Hz.
The code is shown below:

```
#Visualization of frequency response of the FIR filters: we can use the butterworth functions.
for filter_type in ['low', 'high', 'band']:
```

```
    plt.figure(figsize=(10, 6))
    for order in order_values2:
        if filter_type == 'low':
            coefficients = firwin(order + 1, high_cutoff / (0.5 * fs), pass_zero=True)
        elif filter_type == 'high':
            coefficients = firwin(order + 1, low_cutoff / (0.5 * fs), pass_zero=True)
        elif filter_type == 'band':
            coefficients = firwin(order + 1, [low_cutoff / (0.5 * fs), high_cutoff / (0.5 * fs)],
     pass_zero=True)

        #Frequency response based on filter type
        w, h = freqz(coefficients, worN=8000)

        plt.plot((fs * 0.5 / np.pi) * w, abs(h), label=f"Order {order}")
        #w is converted into Heartz from radiants, while abs(h) takes just the amplitude part of
     the magnitude.

    plt.title(f"Frequency Response of FIR {filter_type.capitalize()}-pass Filter")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
    plt.grid(True)
    plt.legend()
    plt.show()
```

Figures 25, 26 and 27 show the Frequency response of the FIR filters. As we can see, as the order increases, the curve becomes steeper in the shift between the passband and the stopband. This is due to the fact that the filter becomes more selective. Moreover, in the Band-pass filter, the "bell" shape of the frequency response becomes more pronounced as the order increases.

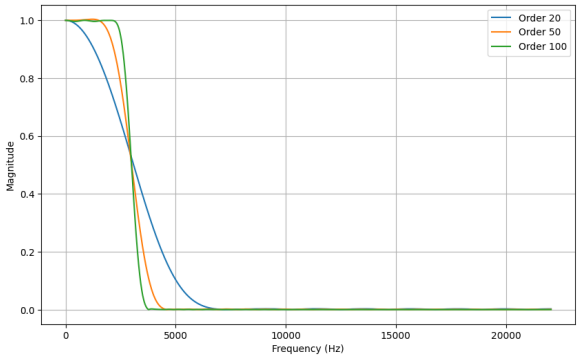

Fig. 25: Frequency Response of FIR Low-pass Filter.

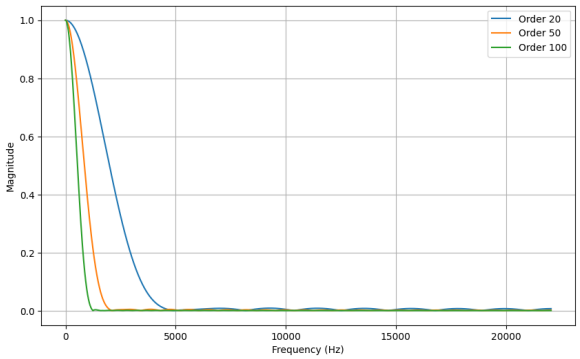

Fig. 26: Frequency Response of FIR High-pass Filter.

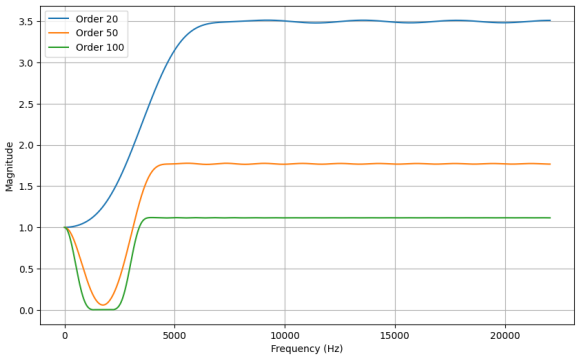

Fig. 27: Frequency Response of FIR Band-pass Filter.

4 FIR vs. IIR Filters

In this section we compare IIR filters and FIR ones calculated in sections 2 and 3.
In order to have a deep comparison, first of all we look at the group delay. It is a measure of delay of frequency components and basically it is the temporal average delay that each components suffers because of the filter. It is useful to see the effect of the filter on the phase of the signal. The phase of a signal represents the relative alignment of each exponential with the oscillation at frequency $(\frac{2\pi}{N})k$ contained in the signal. This alignment determines the shape of the signal in the discrete-time domain. To perform this, we use the function *group_delay* that catches the frequencies in radiant (w), and the group delay in samples (g). This is perform for each filter and for every order. The code is the following one:

```
from scipy.signal import group_delay

#DELAY IIR
fig, axs = plt.subplots(len(order_values), 3, figsize=(18, 5 * len(order_values)))

for i, order in enumerate(order_values):
    #Coefficients IIR Low-pass filter
    b_low, a_low = lowpass_butterworth(high_cutoff, fs, order=order)
    w_iir_low, gd_iir_low = group_delay((b_low, a_low)) #Delay

    #Coefficients IIR High-pass filter
    b_high, a_high = highpass_butterworth(low_cutoff, fs, order=order)
    w_iir_high, gd_iir_high = group_delay((b_high, a_high)) #Delay

    #Coefficients IIR Band-pass filter
    b_band, a_band = bandpass_butterworth(low_cutoff, high_cutoff, fs, order=order)
    w_iir_band, gd_iir_band = group_delay((b_band, a_band)) #Delay

    #Plot IIR Low-pass filter
    axs[i, 0].plot(w_iir_low * fs / (2 * np.pi), gd_iir_low, color="orchid")
    axs[i, 0].set_title(f"IIR Low-pass (Order {order})")
    axs[i, 0].set_xlabel("Frequency (Hz)")
    axs[i, 0].set_ylabel("Group Delay (samples)")
    axs[i, 0].grid(True)

    #Plot IIR High-pass filter
    axs[i, 1].plot(w_iir_high * fs / (2 * np.pi), gd_iir_high, color="darkorange")
    axs[i, 1].set_title(f"IIR High-pass (Order {order})")
    axs[i, 1].set_xlabel("Frequency (Hz)")
    axs[i, 1].set_ylabel("Group Delay (samples)")
    axs[i, 1].grid(True)

    #Plot IIR Band-pass filter
    axs[i, 2].plot(w_iir_band * fs / (2 * np.pi), gd_iir_band, color="seagreen")
    axs[i, 2].set_title(f"IIR Band-pass (Order {order})")
    axs[i, 2].set_xlabel("Frequency (Hz)")
    axs[i, 2].set_ylabel("Group Delay (samples)")
    axs[i, 2].grid(True)

#Space between subplots
plt.subplots_adjust(hspace=0.4, wspace=0.4)
plt.tight_layout()
plt.show()
```

Figure 28 shows the Delay of IIR filters. In thsi case we can see significant variations. Moreover, the group delay is not linear, and this means that the components in frequency are delayed in different times. This could cause distortions. The increase of the order is sometimes too elevated and causes many variations, as it is evident from the IIR Band-pass filter in order=6.

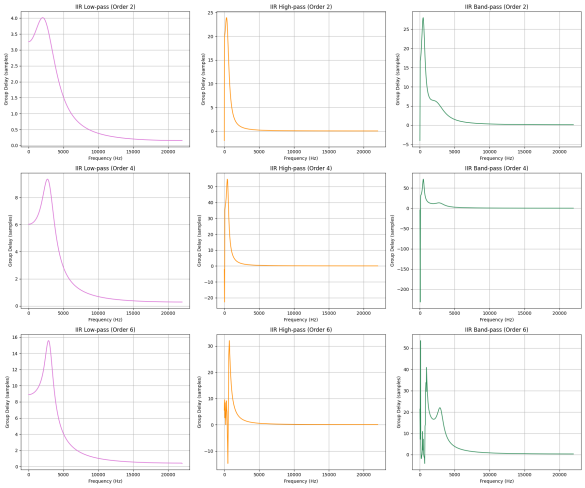

Fig. 28: Delay of IIR filters for order=[2, 4, 6].

Same process also for FIR filters. The code is shown below:

```
#DELAY FIR
fig, axs = plt.subplots(len(order_values2), 3, figsize=(18, 5 * len(order_values2)))

for i, order in enumerate(order_values2):
    #Coefficients FIR Low-pass filter
    coefficients_low = firwin(order + 1, high_cutoff / (0.5 * fs), pass_zero=True)
    w_fir_low, gd_fir_low = group_delay((coefficients_low, 1)) #Delay

    #Coefficients FIR High-pass filter
    coefficients_high = firwin(order + 1, low_cutoff / (0.5 * fs), pass_zero=False)
    w_fir_high, gd_fir_high = group_delay((coefficients_high, 1)) #Delay

    #Coefficients FIR Band-pass filter
    coefficients_band = firwin(order + 1, [low_cutoff / (0.5 * fs), high_cutoff / (0.5 * fs)],
     pass_zero=False)
    w_fir_band, gd_fir_band = group_delay((coefficients_band, 1)) #Delay

    #Plot FIR Low-pass filter
    axs[i, 0].plot(w_fir_low * fs / (2 * np.pi), gd_fir_low, color="orchid")
    axs[i, 0].set_title(f"FIR Low-pass (Order {order})")
    axs[i, 0].set_xlabel("Frequency (Hz)")
    axs[i, 0].set_ylabel("Group Delay (samples)")
    axs[i, 0].grid(True)

    #Plot FIR High-pass filter
    axs[i, 1].plot(w_fir_high * fs / (2 * np.pi), gd_fir_high, color="darkorange")
    axs[i, 1].set_title(f"FIR High-pass (Order {order})")
    axs[i, 1].set_xlabel("Frequency (Hz)")
    axs[i, 1].set_ylabel("Group Delay (samples)")
    axs[i, 1].grid(True)

    #Plot FIR Band-pass filter
    axs[i, 2].plot(w_fir_band * fs / (2 * np.pi), gd_fir_band, color="seagreen")
    axs[i, 2].set_title(f"FIR Band-pass (Order {order})")
    axs[i, 2].set_xlabel("Frequency (Hz)")
    axs[i, 2].set_ylabel("Group Delay (samples)")
    axs[i, 2].grid(True)

#Space between subplots
plt.subplots_adjust(hspace=0.4, wspace=0.4)
plt.tight_layout()
plt.show()
```

Figure 29 shows the delay of FIR filters. As we can see, in all cases we have an almost constant group delay. This means that all the frequency components are delayed at the same time, differently from IIR cases. So in this sense, we have linearity with FIR filters. Moreover, in general when we increase the order we have a higher absolute value of delay, since a filter with a higher order has a longer impulse response. However, in this case and differently from IIR filters, this aspect doesn't cause huge differences from the lower ones.

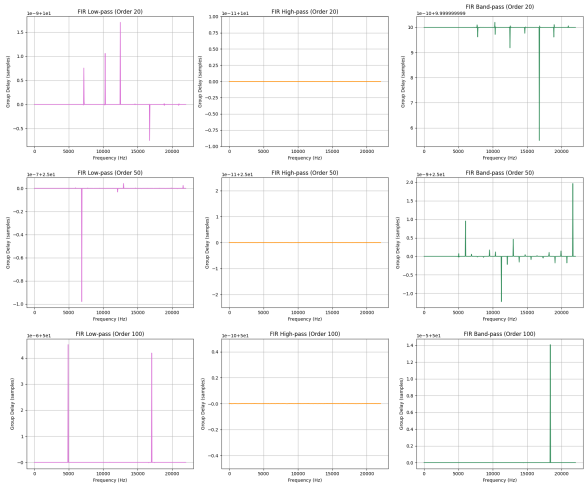

Fig. 29: Delay of FIR filters for order=[20, 50, 100].

After looking at the delay, we can calculate the Ripples of the Passband and the Stopband. Ripples are like oscillations. The passband ripple is calculated as the maximum deviation of the response in decibel. This is performed through the difference between the maximum and the minimum value of the response. This measures gives us an idea of how much the filter is stable inside the band.
Regarding the stopband, it is calculated as the maximum value of the response outside the band. It indicates the efficiency of the filter to attenuate undesirable frequencies.
Finally, we also calculate the sharpness of the transition band as the slope of the linear regression, calculated around an interval (arbitrary) around cutoff frequencies. A higher value of slope indicates a better optimization of the filter.
The code for the IIR filters is shown below:

```
from scipy.stats import linregress

#RIPPLE IIR
iir_pass_low = []
iir_pass_high = []
iir_pass_band = []

iir_stop_low = []
iir_stop_high = []
iir_stop_band = []

iir_sharp_low = []
iir_sharp_high = []
iir_sharp_band = []

for i, order in enumerate(order_values):

    #1) Coefficients IIR Low-pass filter
    b_low, a_low = lowpass_butterworth(high_cutoff, fs, order=order)
    w_iir_low, h_iir_low = freqz(b_low, a_low, worN=8000) #Ripple
    #Passband ripple calculation: Max. deviation of the response in the band
    iir_passband_resp_low =  abs(h_iir_low[w_iir_low <= high_cutoff * 2 * np.pi / fs]) #Response
     in the band
    iir_passband_ripple_low = 20 * np.log10(np.max(iir_passband_resp_low)) - 20 * np.log10(np.min(
     iir_passband_resp_low))
    iir_pass_low.append(iir_passband_ripple_low)

```

```
    #Stopband ripple calculation: max. value of the response in the stop band in dB
    iir_stopband_resp_low = abs(h_iir_low[w_iir_low > high_cutoff * 2 * np.pi / fs])
    iir_stopband_ripple_low = 20 * np.log10(np.max(iir_stopband_resp_low))
    iir_stop_low.append(iir_stopband_ripple_low)

    #Sharpness of the transition band
    freq_Hz_iir_low = fs * 0.5 * w_iir_low / np.pi #Frequencies in Hz
    magn_dB_iir_low = 20 * np.log10(np.abs(h_iir_low)) #Magnitude in decibel
    #Cutoff frequency interval for the regression
    cutoff_index = np.where(freq_Hz_iir_low >= high_cutoff)[0][0]  #Finding the first index that exceeds the cutoff
    transition_indices = np.arange(cutoff_index - 20, cutoff_index + 20)  #Selecting some points around the cutoff frequency to construct the linear regression
    #Linear regression
    slope_low, intercept_low, r_value, p_value, std_err = linregress(freq_Hz_iir_low[transition_indices], magn_dB_iir_low[transition_indices])
    iir_sharp_low.append(slope_low)

    #2) Coefficients IIR High-pass filter
    b_high, a_high = highpass_butterworth(low_cutoff, fs, order=order)
    w_iir_high, h_iir_high = freqz(b_high, a_high, worN=8000) #Ripple
    #Passband ripple calculation: Max. deviation of the response in the band
    iir_passband_resp_high =  abs(h_iir_high[w_iir_high >= low_cutoff * 2 * np.pi / fs]) #Response in the band
    iir_passband_ripple_high = 20 * np.log10(np.max(iir_passband_resp_high)) - 20 * np.log10(np.min(iir_passband_resp_high))
    iir_pass_high.append(iir_passband_ripple_high)

    #Stopband ripple calculation: max. value of the response in the stop band in dB
    iir_stopband_resp_high = abs(h_iir_high[w_iir_high < low_cutoff * 2 * np.pi / fs])
    iir_stopband_ripple_high = 20 * np.log10(np.max(iir_stopband_resp_high))
    iir_stop_high.append(iir_stopband_ripple_high)

    #Sharpness of the transition band
    freq_Hz_iir_high = fs * 0.5 * w_iir_high / np.pi #Frequencies in Hz
    magn_dB_iir_high = 20 * np.log10(np.abs(h_iir_high)) #Magnitude in decibel
    #Cutoff frequency interval for the regression
    cutoff_index = np.where(freq_Hz_iir_high <= low_cutoff)[0][-1]  #Finding the first index that exceeds the cutoff
    transition_indices = np.arange(cutoff_index - 20, cutoff_index + 20)  #Selecting some points around the cutoff frequency to construct the linear regression
    #Linear regression
    slope_high, intercept_high, r_value, p_value, std_err = linregress(freq_Hz_iir_high[transition_indices], magn_dB_iir_high[transition_indices])
    iir_sharp_high.append(slope_high)

    #3) Coefficients IIR Band-pass filter
    b_band, a_band = bandpass_butterworth(low_cutoff, high_cutoff, fs, order=order)
    w_iir_band, h_iir_band = freqz(b_band, a_band, worN=8000) #Ripple
    #Passband ripple calculation: Max. deviation of the response in the band
    iir_passband_resp_band = abs(h_iir_band[(w_iir_band >= low_cutoff * 2 * np.pi / fs) & (w_iir_band <= high_cutoff * 2 * np.pi / fs)])
    iir_passband_ripple_band = 20 * np.log10(np.max(iir_passband_resp_band)) - 20 * np.log10(np.min(iir_passband_resp_band))
    iir_pass_band.append(iir_passband_ripple_band)

    #Stopband ripple calculation: max response in stopbands (below low_cutoff and above high_cutoff)
    iir_stopband_resp_band = np.concatenate([
        abs(h_iir_band[w_iir_band < low_cutoff * 2 * np.pi / fs]),
        abs(h_iir_band[w_iir_band > high_cutoff * 2 * np.pi / fs])
    ])
    iir_stopband_ripple_band = 20 * np.log10(np.max(iir_stopband_resp_band))
    iir_stop_band.append(iir_stopband_ripple_band)

    #Sharpness of the transition band
    freq_Hz_iir_band = fs * 0.5 * w_iir_band / np.pi #Frequencies in Hz
    magn_dB_iir_band = 20 * np.log10(np.abs(h_iir_band)) #Magnitude in decibel
    #Cutoff frequency interval for the regression
    low_cutoff_index = np.where(freq_Hz_iir_band >= low_cutoff)[0][0] #Finding the indeces that exceed the cutoff
    high_cutoff_index = np.where(freq_Hz_iir_band <= high_cutoff)[0][-1]
    transition_indices = np.arange(low_cutoff_index, high_cutoff_index + 1)  #Selecting some points around the cutoff frequency to construct the linear regression
    #Linear regression
    slope_band, intercept_band, r_value, p_value, std_err = linregress(freq_Hz_iir_band[transition_indices], magn_dB_iir_band[transition_indices])
    iir_sharp_band.append(slope_band)
```

We do the same for FIR filters. The code is shown below:

```
#RIPPLE FIR
```

```
fir_pass_low = []
fir_pass_high = []
fir_pass_band = []

fir_stop_low = []
fir_stop_high = []
fir_stop_band = []

fir_sharp_low = []
fir_sharp_high = []
fir_sharp_band = []

for i, order in enumerate(order_values2):

    #1) Coefficients FIR Low-pass filter
    coefficients_low = firwin(order + 1, high_cutoff / (0.5 * fs), pass_zero=True)
    w_fir_low, h_fir_low = freqz(coefficients_low, worN=8000) #Ripple
    #Passband ripple calculation: Max. deviation of the response in the band
    fir_passband_resp_low =  abs(h_fir_low[w_fir_low <= high_cutoff * 2 * np.pi / fs]) #Response in the band
    fir_passband_ripple_low = 20 * np.log10(np.max(fir_passband_resp_low)) - 20 * np.log10(np.min(fir_passband_resp_low))
    fir_pass_low.append(fir_passband_ripple_low)

    #Stopband ripple calculation: max. value of the response in the stop band in dB
    fir_stopband_resp_low = abs(h_fir_low[w_fir_low > high_cutoff * 2 * np.pi / fs])
    fir_stopband_ripple_low = 20 * np.log10(np.max(fir_stopband_resp_low))
    fir_stop_low.append(fir_stopband_ripple_low)

    #Sharpness of the transition band
    freq_Hz_fir_low = fs * 0.5 * w_fir_low / np.pi #Frequencies in Hz
    magn_dB_fir_low = 20 * np.log10(np.abs(h_fir_low)) #Magnitude in decibel
    #Cutoff frequency interval for the regression
    cutoff_index = np.where(freq_Hz_fir_low >= high_cutoff)[0][0]  #Finding the first index that exceeds the cutoff
    transition_indices = np.arange(cutoff_index - 20, cutoff_index + 20)  #Selecting some points around the cutoff frequency to construct the linear regression
    #Linear regression
    slope_low, intercept_low, r_value, p_value, std_err = linregress(freq_Hz_fir_low[transition_indices], magn_dB_fir_low[transition_indices])
    fir_sharp_low.append(slope_low)

    #2) Coefficients FIR High-pass filter
    coefficients_high = firwin(order + 1, low_cutoff / (0.5 * fs), pass_zero=False)
    w_fir_high, h_fir_high = freqz(coefficients_high, worN=8000) #Ripple
    #Passband ripple calculation: Max. deviation of the response in the band
    fir_passband_resp_high =  abs(h_fir_high[w_fir_high >= low_cutoff * 2 * np.pi / fs]) #Response in the band
    fir_passband_ripple_high = 20 * np.log10(np.max(fir_passband_resp_high)) - 20 * np.log10(np.min(fir_passband_resp_high))
    fir_pass_high.append(fir_passband_ripple_high)

    #Stopband ripple calculation: max. value of the response in the stop band in dB
    fir_stopband_resp_high = abs(h_fir_high[w_fir_high < low_cutoff * 2 * np.pi / fs])
    fir_stopband_ripple_high = 20 * np.log10(np.max(fir_stopband_resp_high))
    fir_stop_high.append(fir_passband_ripple_high)

    #Sharpness of the transition band
    freq_Hz_fir_high = fs * 0.5 * w_fir_high / np.pi #Frequencies in Hz
    magn_dB_fir_high = 20 * np.log10(np.abs(h_fir_high)) #Magnitude in decibel
    #Cutoff frequency interval for the regression
    cutoff_index = np.where(freq_Hz_fir_high <= low_cutoff)[0][-1]  #Finding the first index that exceeds the cutoff
    transition_indices = np.arange(cutoff_index - 20, cutoff_index + 20)  #Selecting some points around the cutoff frequency to construct the linear regression
    #Linear regression
    slope_high, intercept_high, r_value, p_value, std_err = linregress(freq_Hz_fir_high[transition_indices], magn_dB_fir_high[transition_indices])
    fir_sharp_high.append(slope_high)

    #3) Coefficients FIR Band-pass filter
    coefficients_band = firwin(order + 1, [low_cutoff / (0.5 * fs), high_cutoff / (0.5 * fs)], pass_zero=False)
    w_fir_band, h_fir_band = freqz(coefficients_band, worN=8000) #Ripple
    #Passband ripple calculation: Max. deviation of the response in the band
    fir_passband_resp_band = abs(h_fir_band[(w_fir_band >= low_cutoff * 2 * np.pi / fs) & (w_fir_band <= high_cutoff * 2 * np.pi / fs)])
    fir_passband_ripple_band = 20 * np.log10(np.max(fir_passband_resp_band)) - 20 * np.log10(np.min(fir_passband_resp_band))
    fir_pass_band.append(fir_passband_ripple_band)

```

```
    #Stopband ripple calculation: max response in stopbands (below low_cutoff and above
     high_cutoff)
    fir_stopband_resp_band = np.concatenate([
        abs(h_fir_band[w_fir_band < low_cutoff * 2 * np.pi / fs]),
        abs(h_fir_band[w_fir_band > high_cutoff * 2 * np.pi / fs])
    ])
    fir_stopband_ripple_band = 20 * np.log10(np.max(fir_stopband_resp_band))
    fir_stop_band.append(fir_stopband_ripple_band)

    #Sharpness of the transition band
    freq_Hz_fir_band = fs * 0.5 * w_fir_band / np.pi #Frequencies in Hz
    magn_dB_fir_band = 20 * np.log10(np.abs(h_fir_band)) #Magnitude in decibel
    #Cutoff frequency interval for the regression
    low_cutoff_index = np.where(freq_Hz_fir_band >= low_cutoff)[0][0] #Finding the indeces that
     exceed the cutoff
    high_cutoff_index = np.where(freq_Hz_fir_band <= high_cutoff)[0][-1]
    transition_indices = np.arange(low_cutoff_index, high_cutoff_index + 1)  #Selecting some
     points around the cutoff frequency to construct the linear regression
    #Linear regression
    slope_band, intercept_band, r_value, p_value, std_err = linregress(freq_Hz_fir_band[
     transition_indices], magn_dB_fir_band[transition_indices])
    fir_sharp_band.append(slope_band)
```

Tables 3, 4, 5, 6 show the results. For IIR filters, the ripples measures for both passband and stopband, has values around 3 dB and between -3.07 dB and -3 dB respectively. As we increase the order, the values of the ripples are lower. Regarding the sharpness, the values increases their absolute value as the order does.
FIR filter have a similar behaviour, but they have higher values of passband ripples (¿ 5 dB) and also of stopband (between -6 and -4 dB). So, FIR filters tend to have more oscillations inside the band. The sharpness ofFIR filter is instead negative for Low-pass filter and Band-pass one. For High-pass filter we have positive values. This suggests that FIR filters have a better precision with higher orders (confirmation of previous analysis).

Order	Low-pass PR	Low-pass SR	High-pass PR	High-pass SR	Band-pass PR	Band-pass SR
2	3.0067	-3.0149	2.9820	-3.0298	3.0053	-3.0168
4	3.0031	-3.0196	2.9538	-3.0494	3.0003	-3.0233
6	2.9996	-3.0243	2.9259	-3.0691	2.9954	-3.0297

Table 3: Passband Ripple (PR) and Stopband Ripple (SR) in decibel for IIR filters at order=[2, 4, 6].

Order	Low-pass Sharpness	High-pass Sharpness	Band-pass Sharpness
20	-0.003	0.0178	-0.0007
50	-0.006	0.0362	-0.0005
100	-0.009	0.0551	-0.0004

Table 4: Sharpness values for IIR filters at order=[2, 4, 6].

Order	Low-pass PR	Low-pass SR	High-pass PR	High-pass SR	Band-pass PR	Band-pass SR
20	5.6074	-5.6192	2.3248	2.3248	3.8295	0.9277
50	6.0176	-6.0145	5.1876	5.1876	5.6610	-4.8998
100	6.0190	-6.0776	6.0344	6.0344	6.0608	-6.0717

Table 5: Passband Ripple (PR) and Stopband Ripple (SR) in decibel for FIR filters at order=[20, 50, 100].

5 Adding Noise and Re-filtering with FIR and IIR Filters

In this section we perform instead a signal adding white Gaussian noise. The process is the one performed also in assignment 4. Here we have a SNR of 20 decibel, and after finding the signal power, we calculate the noise power. Then, the generation of normal random values for noise is trivial and

Order	Low-pass Sharpness	High-pass Sharpness	Band-pass Sharpness
20	-0.0043	0.0004	-0.0015
50	-0.0106	0.0066	-0.0001
100	-0.0214	0.0216	0.0000

Table 6: Sharpness values for FIR filters at order=[20, 50, 100].

is performed through the function *random.normal* (the mean is 0). The resulting speech signal is the sum of the original one and the noise generated. The code to implement this is shown below:

```
#Adding White Gaussian Noise
SNR_dB = 20
#Conversion from dB to linear SNR
SNR_linear = 10**(SNR_dB/10)

#Power
signal_power = np.mean(speech_signal**2)
#Noise power
noise_power = signal_power/SNR_linear
#Generating noise
noise = np.random.normal(0, np.sqrt(noise_power), speech_signal.shape)

#Adding noise to the speech signal
signal_noise = speech_signal + noise
```

Figure 30 shows the signal with noise obtained.

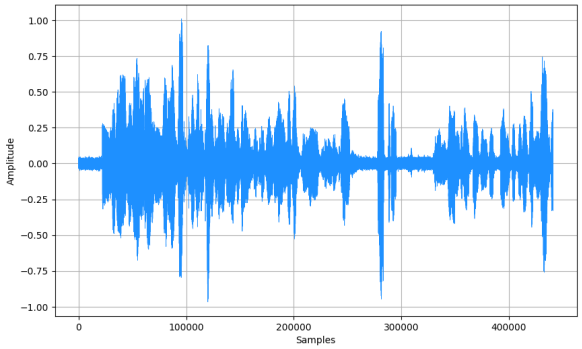

Fig. 30: Signal with White Gaussian Noise.

Again, we perform the same analysis done in section 1, finding the spectrum through the FFT and its frequencies. The code is shown below:

```
#Applying the FFT
spectrum_noise = fft(signal_noise) #Spectrum of the signal with noise

#We take the correspondent frequencies of the samples of the Fourier transform
freq_FFT_noise = fftfreq(len(signal_noise), 1/fs)
#We take just the half positive part of the spectrum's frequencies since they are simmetric to 0
    for real signals
freq_FFT_noise = freq_FFT_noise[: len(signal_noise)//2]

#Magnitude
magnitude_noise = np.abs(spectrum_noise[: len(signal_noise) // 2])
#Transforming the magnitude in decibel
magnitude_dB_noise = 20*np.log10(magnitude_noise)
```

Figures 31 and 32 shown the Frequency spectrum of the noisy signal. In particular from the second graph, we can see the noise added since the energy is concentrated in a bigger band.

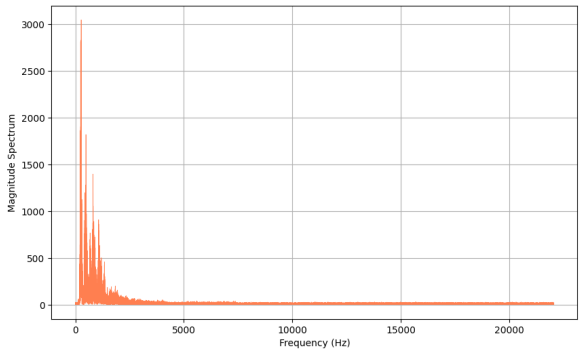

Fig. 31: Frequency Spectrum of the signal with noise.

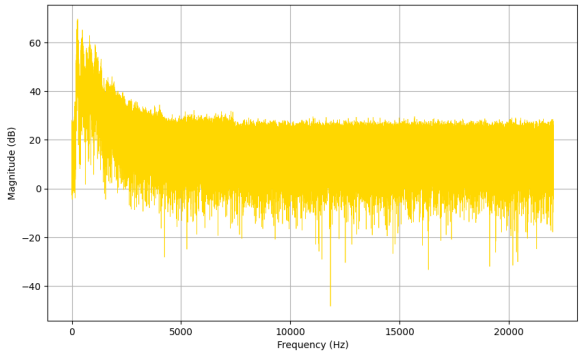

Fig. 32: Frequency Spectrum (dB) of the signal with noise.

After this preliminary analysis, we apply the IIR filters using always the same orders of section 2. Below the code for Low-pass filter:

```
#Application of the IIR filters to the signal with noise:
#IIR Low-pass filtered speech signal

for order in order_values:
    low_filtered_noise = IIR_lowpass_filter(signal_noise, high_cutoff, fs, order)

    #Duration
    duration_low_noise = len(low_filtered_noise) / fs

    #Spectrum
    spectrum_low_noise = fft(low_filtered_noise)

    #Frequencies of the FFT
    freq_FFT_low_noise = fftfreq(len(low_filtered_noise), 1/fs)
    freq_FFT_low_noise = freq_FFT_low[: len(low_filtered_noise)//2]

    #Magnitude
    magnitude_low_noise = np.abs(spectrum_low_noise[: len(low_filtered_noise) // 2])
    #Transforming the magnitude in decibel
    magnitude_low_dB_noise = 20*np.log10(magnitude_low_noise)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_low_noise, len(low_filtered_noise)), low_filtered_noise,
     color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"IIR Low-pass filtered Signal with noise (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
```

```
    axs[1].plot(freq_FFT_low_noise, magnitude_low_noise, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_low_noise, magnitude_low_dB_noise, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the IIR Low-pass filtered signal with noise (
    Order {order})", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 33, 34, 35 show the outputs of the application of the IIR Low-pass filters to the noisy signal for each order. Also in this case, the higher the order, the better the selection.

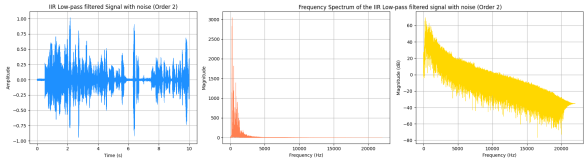

Fig. 33: Plots of IIR Low-pass filtered noisy signal, Order=2.

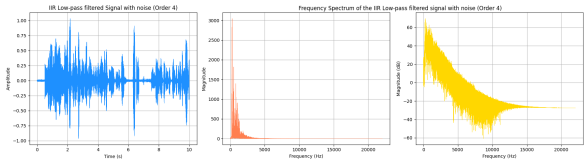

Fig. 34: Plots of IIR Low-pass filtered noisy signal, Order=4.

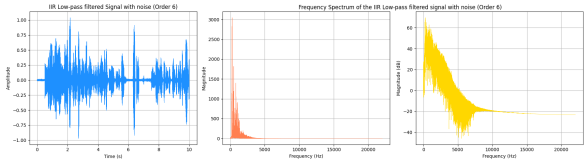

Fig. 35: Plots of IIR Low-pass filtered noisy signal, Order=6.

Below there is the code for IIR High-pass filter to noisy signal:

```
#IIR High-pass filtered speech signal
for order in order_values:
    high_filtered_noise = IIR_highpass_filter(signal_noise, low_cutoff, fs, order)

    #Duration
    duration_high_noise = len(high_filtered_noise) / fs

    #Spectrum
    spectrum_high_noise = fft(high_filtered_noise)

    #Frequencies of the FFT
    freq_FFT_high_noise = fftfreq(len(high_filtered_noise), 1/fs)
    freq_FFT_high_noise = freq_FFT_high[: len(high_filtered_noise)//2]
```

```
14
15      #Magnitude
16      magnitude_high_noise = np.abs(spectrum_high_noise[: len(high_filtered_noise) // 2])
17
18      #Transforming the magnitude in decibel
19      magnitude_high_dB_noise = 20*np.log10(magnitude_high_noise)
20
21      #Plot
22      fig, axs = plt.subplots(1, 3, figsize=(18, 5))
23
24      #Plot time domain
25      axs[0].plot(np.linspace(0, duration_high_noise, len(high_filtered_noise)), high_filtered_noise
        , color='dodgerblue', linewidth=0.5)
26      axs[0].set_title(f"IIR High-pass filtered Signal with noise (Order {order})")
27      axs[0].set_xlabel("Time (s)")
28      axs[0].set_ylabel("Amplitude")
29      axs[0].grid(True)
30
31      #Plot Magnitude Spectrum
32      axs[1].plot(freq_FFT_high_noise, magnitude_high_noise, color="coral", linewidth=0.5)
33      axs[1].set_xlabel("Frequency (Hz)")
34      axs[1].set_ylabel("Magnitude")
35      axs[1].grid(True)
36
37      #Plot Magnitude in dB
38      axs[2].plot(freq_FFT_high_noise, magnitude_high_dB_noise, color="gold", linewidth=0.5)
39      axs[2].set_xlabel("Frequency (Hz)")
40      axs[2].set_ylabel("Magnitude (dB)")
41      axs[2].grid(True)
42
43      #Title of the second and third graphs
44      fig.text(0.68, 0.94, f"Frequency Spectrum of the IIR High-pass filtered signal with noise (
        Order {order})", ha='center', fontsize=12, color="black")
45
46      #Space between subplots
47      plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
48      plt.tight_layout()
49      plt.show()
```

Figures 36, 37, 38 show the outputs of the application of the IIR High-pass filters to the noisy signal for each order. Here we can't see evident differences varying the order. This suggests that the noise was concentrated in higher frequencies that are maintained, with this filter.

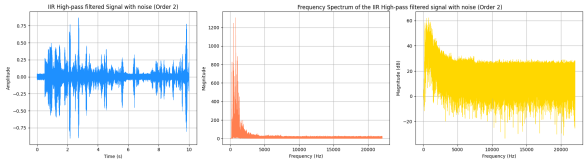

Fig. 36: Plots of IIR High-pass filtered noisy signal, Order=2.

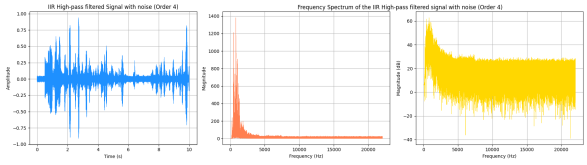

Fig. 37: Plots of IIR High-pass filtered noisy signal, Order=4.

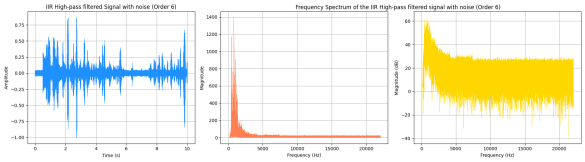

Fig. 38: Plots of IIR High-pass filtered noisy signal, Order=6.

Below there is the code for the IIR Band-pass filter to the noisy signal:

```
#IIR Band-pass filtered speech signal with noise
for order in order_values:
    band_filtered_noise = IIR_bandpass_filter(signal_noise, low_cutoff, high_cutoff, fs, order)

    #Duration
    duration_band_noise = len(band_filtered_noise) / fs

    #Spectrum
    spectrum_band_noise = fft(band_filtered_noise)

    #Frequencies of the FFT
    freq_FFT_band_noise = fftfreq(len(band_filtered_noise), 1/fs)
    freq_FFT_band_noise = freq_FFT_band[: len(band_filtered_noise)//2]

    #Magnitude
    magnitude_band_noise = np.abs(spectrum_band_noise[: len(band_filtered_noise) // 2])
    #Transforming the magnitude in decibel
    magnitude_band_dB_noise = 20*np.log10(magnitude_band_noise)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_band_noise, len(band_filtered_noise)), band_filtered_noise
    , color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"IIR Band-pass filtered Signal with noise (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_band_noise, magnitude_band_noise, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_band_noise, magnitude_band_dB_noise, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the IIR Band-pass filtered signal with noise (
    Order {order})", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures 39, 40, 41 show the outputs of the application of the IIR Band-pass filters to the noisy signal for each order. The filtration looks also better than the IIR Low-pass filter.

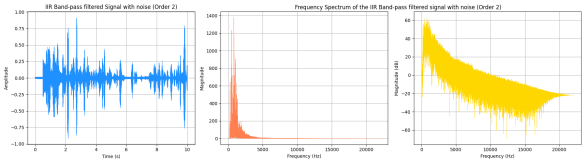

Fig. 39: Plots of IIR Band-pass filtered noisy signal, Order=2.

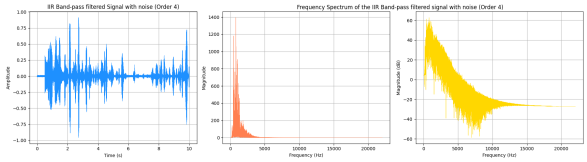

Fig. 40: Plots of IIR Band-pass filtered noisy signal, Order=4.

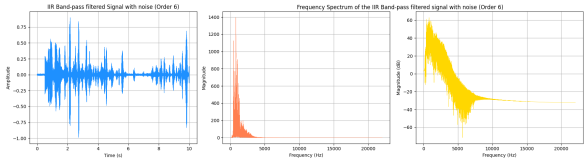

Fig. 41: Plots of IIR Band-pass filtered noisy signal, Order=6.

Now we can apply the FIR filters, same process of section 3. We start with Low-pass filter:

```
#FIR Low-pass filtered speech signal with noise
for order in order_values2:
    low_filtered_noise = FIR_lowpass_filter(signal_noise, high_cutoff, fs, order)

    #Duration
    duration_low_noise = len(low_filtered_noise) / fs

    #Spectrum
    spectrum_low_noise = fft(low_filtered_noise)

    #Frequencies of the FFT
    freq_FFT_low_noise = fftfreq(len(low_filtered_noise), 1/fs)
    freq_FFT_low_noise = freq_FFT_low_noise[: len(low_filtered_noise)//2]

    #Magnitude
    magnitude_low_noise = np.abs(spectrum_low_noise[: len(low_filtered_noise) // 2])

    #Transforming the magnitude in decibel
    magnitude_low_dB_noise = 20*np.log10(magnitude_low_noise)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_low_noise, len(low_filtered_noise)), low_filtered_noise,
     color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"FIR Low-pass filtered Signal with noise (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_low_noise, magnitude_low_noise, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_low_noise, magnitude_low_dB_noise, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the FIR Low-pass filtered signal with noise (
     Order {order})", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures **??**, **??**, **??** show the outputs of the application of the FIR Low-pass filters to the noisy signal for each order. The band is narrower than the one of IIR filters, and the spectrum seems more linear.

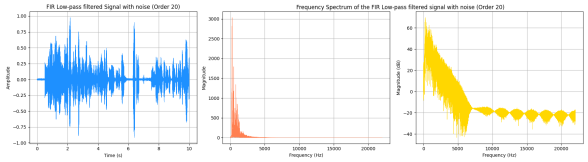

Fig. 42: Plots of FIR Low-pass filtered noisy signal, Order=20.

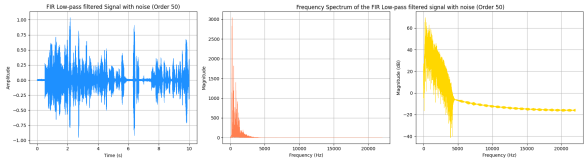

Fig. 43: Plots of FIR Low-pass filtered noisy signal, Order=50.

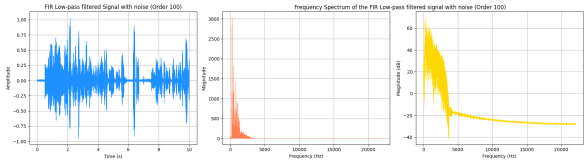

Fig. 44: Plots of FIR Low-pass filtered noisy signal, Order=100.

Below the code for the application fo the FIR High-pass filter to the noisy signal:

```
#FIR High-pass filtered speech signal with noise
for order in order_values2:
    high_filtered_noise = FIR_highpass_filter(signal_noise, low_cutoff, fs, order)

    #Duration
    duration_high_noise = len(high_filtered_noise) / fs

    #Spectrum
    spectrum_high_noise = fft(high_filtered_noise)

    #Frequencies of the FFT
    freq_FFT_high_noise = fftfreq(len(high_filtered_noise), 1/fs)
    freq_FFT_high_noise = freq_FFT_high_noise[: len(high_filtered_noise)//2]

    #Magnitude
    magnitude_high_noise = np.abs(spectrum_high_noise[: len(high_filtered_noise) // 2])

    #Transforming the magnitude in decibel
    magnitude_high_dB_noise = 20*np.log10(magnitude_high_noise)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_high_noise, len(high_filtered_noise)), high_filtered_noise
    , color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"FIR High-pass filtered Signal with noise (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
```

```
    axs[1].plot(freq_FFT_high_noise, magnitude_high_noise, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_high_noise, magnitude_high_dB_noise, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the FIR High-pass filtered signal with noise (
    Order {order})", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures **??**, **??**, **??** show the outputs of the application of the FIR High-pass filters to the noisy signal for each order. Here, compared to the previous case of Low-pass filter, the band is very narrower, especially with the order equal to 100.

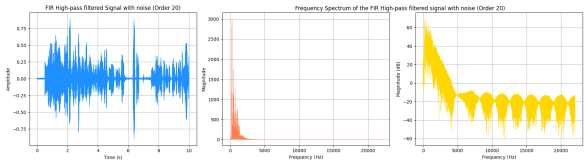

Fig. 45: Plots of FIR High-pass filtered noisy signal, Order=20.

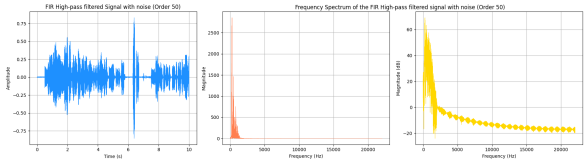

Fig. 46: Plots of FIR High-pass filtered noisy signal, Order=50.

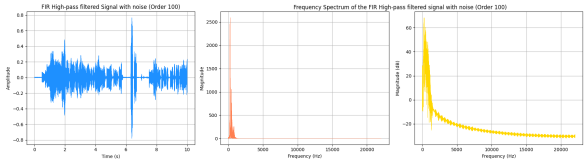

Fig. 47: Plots of FIR High-pass filtered noisy signal, Order=100.

Below the code for the application of the FIR Band-pass filter to the noisy signal:

```
#FIR Band-pass filtered speech signal
for order in order_values2:
    band_filtered_noise = FIR_bandpass_filter(signal_noise, low_cutoff, high_cutoff, fs, order)

    #Duration
    duration_band_noise = len(band_filtered_noise) / fs

    #Spectrum
    spectrum_band_noise = fft(band_filtered_noise)

    #Frequencies of the FFT
    freq_FFT_band_noise = fftfreq(len(band_filtered_noise), 1/fs)
```

```
    freq_FFT_band_noise = freq_FFT_band_noise[: len(band_filtered_noise)//2]

    #Magnitude
    magnitude_band_noise = np.abs(spectrum_band_noise[: len(band_filtered_noise) // 2])

    #Transforming the magnitude in decibel
    magnitude_band_dB_noise = 20*np.log10(magnitude_band_noise)

    #Plot
    fig, axs = plt.subplots(1, 3, figsize=(18, 5))

    #Plot time domain
    axs[0].plot(np.linspace(0, duration_band_noise, len(band_filtered_noise)), band_filtered_noise
    , color='dodgerblue', linewidth=0.5)
    axs[0].set_title(f"FIR Band-pass filtered Signal with noise (Order {order})")
    axs[0].set_xlabel("Time (s)")
    axs[0].set_ylabel("Amplitude")
    axs[0].grid(True)

    #Plot Magnitude Spectrum
    axs[1].plot(freq_FFT_band_noise, magnitude_band_noise, color="coral", linewidth=0.5)
    axs[1].set_xlabel("Frequency (Hz)")
    axs[1].set_ylabel("Magnitude")
    axs[1].grid(True)

    #Plot Magnitude in dB
    axs[2].plot(freq_FFT_band_noise, magnitude_band_dB_noise, color="gold", linewidth=0.5)
    axs[2].set_xlabel("Frequency (Hz)")
    axs[2].set_ylabel("Magnitude (dB)")
    axs[2].grid(True)

    #Title of the second and third graphs
    fig.text(0.68, 0.94, f"Frequency Spectrum of the FIR Band-pass filtered signal with noise (
    Order {order})", ha='center', fontsize=12, color="black")

    #Space between subplots
    plt.subplots_adjust(wspace=0.4) #Space within the graphs of a subplot
    plt.tight_layout()
    plt.show()
```

Figures **??**, **??**, **??** show the outputs of the application of the FIR Band-pass filters to the noisy signal for each order. In this case, Band-pass filter doesn't seem to perform very well as the High-pass one of the previous case, since the band is larger and the noise is not capture totally.

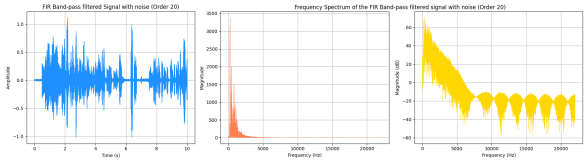

Fig. 48: Plots of FIR Band-pass filtered noisy signal, Order=20.

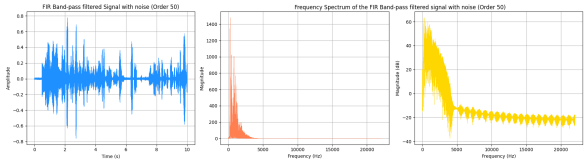

Fig. 49: Plots of FIR Band-pass filtered noisy signal, Order=50.

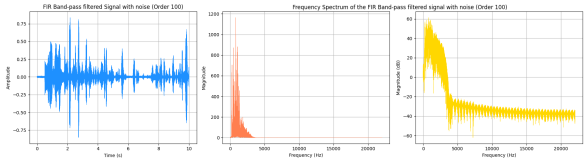

Fig. 50: Plots of FIR Band-pass filtered noisy signal, Order=100.

6 Evaluation and Analysis

For a final evaluation we calculate the Signal-to-Noise Ratio (SNR) for all the cases of IIR and FIR filtering in the clean signal, i.e. the original speech, and also for the noisy signal and all the cases with the filtration performed on this one.
The SNR in the clean signal is calculated with the alternative form of

$$SNR = \frac{Energy}{Variance},$$

since in this case we didn't add noise.
In the case of the noisy signal, we use the usual formula:

$$SNR = \frac{\text{Signal power}}{\text{Noise power}}.$$

In both cases we perform the conversion of the SNR in decibel.

The code for the cases regarding the clean signal is shown below:

```
#SNR for the clean signal without noise
SNR_ecg = 10 * np.log10(np.sum(speech_signal**2)/np.var(speech_signal))

#SNR for the original signal filtered with IIR filters

SNR_ecg_IIR_lowpass = []
SNR_ecg_IIR_highpass = []
SNR_ecg_IIR_bandpass = []

for order in order_values:

    #IIR Low-pass filter
    low_filtered_speech = IIR_lowpass_filter(speech_signal, high_cutoff, fs, order)
    snr_low = 10 * np.log10(np.sum(low_filtered_speech**2)/np.var(low_filtered_speech))
    SNR_ecg_IIR_lowpass.append(snr_low)

    #IIR High-pass filter
    high_filtered_speech = IIR_highpass_filter(speech_signal, low_cutoff, fs, order)
    snr_high = 10 * np.log10(np.sum(high_filtered_speech**2)/np.var(high_filtered_speech))
    SNR_ecg_IIR_highpass.append(snr_high)

    #IIR Band-pass filter
    band_filtered_speech = IIR_bandpass_filter(speech_signal, low_cutoff, high_cutoff, fs, order)
    snr_band = 10 * np.log10(np.sum(band_filtered_speech**2)/np.var(band_filtered_speech))
    SNR_ecg_IIR_bandpass.append(snr_band)

#SNR for the original signal filtered with FIR filters

SNR_ecg_FIR_lowpass = []
SNR_ecg_FIR_highpass = []
SNR_ecg_FIR_bandpass = []

for order in order_values2:

    #FIR Low-pass filter
    low_filtered_speech = FIR_lowpass_filter(speech_signal, high_cutoff, fs, order)
    snr_low = 10 * np.log10(np.sum(low_filtered_speech**2)/np.var(low_filtered_speech))
    SNR_ecg_FIR_lowpass.append(snr_low)

    #FIR High-pass filter
    high_filtered_speech = FIR_highpass_filter(speech_signal, low_cutoff, fs, order)
```

```
    snr_high = 10 * np.log10(np.sum(high_filtered_speech**2)/np.var(high_filtered_speech))
    SNR_ecg_FIR_highpass.append(snr_high)

    #FIR Band-pass filter
    band_filtered_speech = FIR_bandpass_filter(speech_signal, low_cutoff, high_cutoff, fs, order)
    snr_band = 10 * np.log10(np.sum(band_filtered_speech**2)/np.var(band_filtered_speech))
    SNR_ecg_FIR_bandpass.append(snr_band)
```

The Figure 51 shows the results for the clean signal. As we can see, the level of SNR for the clean signal is quite high and this suggests that the quality of the signal is good. In general for both IIR and FIR filters, we have very low differences and the signal remains almost unchanged.

```
SNR clean signal: 56.444387435913086
SNR IIR Low-Pass clean signal: [56.44438594548997, 56.44438594264084, 56.44438593792494]
SNR IIR High-Pass clean signal: [56.44438589813809, 56.4443859061432, 56.444385902228255]
SNR IIR Band-Pass clean signal: [56.44438589893472, 56.44438590263486, 56.44438590093114]
SNR FIR Low-Pass clean signal: [56.444385893682879, 56.444385899365315, 56.44438591945837]
SNR FIR High-Pass clean signal: [56.44438593853137, 56.44438590090352, 56.44438591465792]
SNR FIR Band-Pass clean signal: [56.44438593623682, 56.44438589783132, 56.4443859114386]
```

Fig. 51: Output of the SNR for the clean signal.

The code for the cases regarding the noisy signal is shown below:

```
#Function to calculate the Signal-to-Noise Ratio
def snr_fun(noisy_signal, noise_power):
    signal_power = np.mean(noisy_signal**2)
    snr_value = 10 * np.log10(signal_power / noise_power)
    return snr_value

#SNR for the noisy signal

SNR_noise = snr_fun(signal_noise, noise_power)

#SNR for the noisy signal filtered with IIR filters

SNR_noisy_IIR_lowpass = []
SNR_noisy_IIR_highpass = []
SNR_noisy_IIR_bandpass = []

for order in order_values2:

    #IIR Low-pass filter
    low_filtered_noisy = IIR_lowpass_filter(signal_noise, high_cutoff, fs, order)
    snr_low = snr_fun(low_filtered_noise, noise_power)
    SNR_noisy_IIR_lowpass.append(snr_low)

    #IIR High-pass filter
    high_filtered_noisy = IIR_highpass_filter(signal_noise, low_cutoff, fs, order)
    snr_high = snr_fun(high_filtered_noise, noise_power)
    SNR_noisy_IIR_highpass.append(snr_high)

    #IIR Band-pass filter
    band_filtered_noisy = IIR_bandpass_filter(signal_noise, low_cutoff, high_cutoff, fs, order)
    snr_band = snr_fun(band_filtered_noise, noise_power)
    SNR_noisy_IIR_bandpass.append(snr_band)

#SNR for the original signal filtered with FIR filters

SNR_noisy_FIR_lowpass = []
SNR_noisy_FIR_highpass = []
SNR_noisy_FIR_bandpass = []

for order in order_values2:

    #FIR Low-pass filter
    low_filtered_noisy = FIR_lowpass_filter(signal_noise, high_cutoff, fs, order)
    snr_low = snr_fun(low_filtered_noise, noise_power)
    SNR_noisy_FIR_lowpass.append(snr_low)

    #FIR High-pass filter
    high_filtered_noisy = FIR_highpass_filter(signal_noise, low_cutoff, fs, order)
    snr_high = snr_fun(high_filtered_noise, noise_power)
    SNR_noisy_FIR_highpass.append(snr_high)

    #FIR Band-pass filter
    band_filtered_noisy = FIR_bandpass_filter(signal_noise, low_cutoff, high_cutoff, fs, order)
    snr_band = snr_fun(band_filtered_noise, noise_power)
    SNR_noisy_FIR_bandpass.append(snr_band)
```

The Figure 51 shows the results for the noisy signal. Regarding this case, we can see instead lower value of SNR compared to the clean case. The presence of noise reduces a lot the quality of the sgnal, and the best filters seem to be the Low-pass ones that manage to catch noise enough.

Finally, about the choice of a IIR and a FIR filter, we can say that it depends from the context. When we need linearity FIR filters are preferable, when instead we need more selection with less coefficients, IIR are better.
From our analysis, at the end FIR filters (in particular Low-pass ones) seem to be more linear and more efficient, especially since they have a constant group delay. However, the implementation of them can be computationally expensive.

```
SNR noisy signal: 20.040905589163994
SNR IIR Low-Pass noisy signal: [19.971267300560488, 19.971267300560488, 19.971267300560488]
SNR IIR High-Pass noisy signal: [16.495755146934023, 16.495755146934023, 16.495755146934023]
SNR IIR Band-Pass noisy signal: [14.940101164110324, 14.940101164110324, 14.940101164110324]
SNR FIR Low-Pass noisy signal: [19.971267300560488, 19.971267300560488, 19.971267300560488]
SNR FIR High-Pass noisy signal: [16.495755146934023, 16.495755146934023, 16.495755146934023]
SNR FIR Band-Pass noisy signal: [14.940101164110324, 14.940101164110324, 14.940101164110324]
```

Fig. 52: Output of the SNR for the noisy signal.

References

[1] Stojanovic, Z.: Generate signals with a particular variance and SNR. https://dsp.stackexchange.com/questions/59690/generate-signals-with-a-particular-variance-and-snr (2024)

[2] AdvSolned: Difference Between IIR and FIR Filters: A Practical Design Guide. https://www.advsolned.com/difference-between-iir-and-fir-filters-a-practical-design-guide/

[3] Vaidyanathan, P.P.: Lecture 8: FIR and IIR Filters. https://ocw.mit.edu/courses/6-341-discrete-time-signal-processing-fall-2005/51e3beff8c8ce2289ba292fcdb0040f4_lec08.pdf

[4] u/Doctor-Without-A-Plan: What Are FIR and IIR Filters? Which One to Choose? https://www.reddit.com/r/DSP/comments/x1udgb/what_are_fir_and_iir_filters_which_one_to_choose/

[5] contributors, W.: Signal-to-noise ratio — Wikipedia, The Free Encyclopedia (2023). https://en.wikipedia.org/wiki/Signal-to-noise_ratio

[6] Nait-Ali, A.: Correlation in Digital Signal and Image Processing. Lecture slides, Université Paris-Est Crétéil (UPEC) - Faculté des Sciences et Technologie (2024)

[7] Nait-Ali, A.: Session: Digital Signal Processing. Lecture slides, Université Paris-Est Crétéil (UPEC) - Faculté des Sciences et Technologie (2024)