Speech Processing in Frequency domain Using Periodogram

Sofia Noemi Crobeddu
student number: 032410554
International Master of Biometrics and Intelligent Vision - Université Paris-Est Cretéil (UPEC)
Assignment 2

1 Introduction

Speech signal processing is a the study of the human voice. This work focuses on the analysis of a personal recorded vocal signal, analyzing the spectrum, the frequency and using processing techniques, such as sampling frequency, windowing, and the length of the signal. Special attention is given to the Power Spectral Density (PSD) analysis.

2 Speech Signal Acquisition process

For the recording of the speech, I used the library *sounddevice* of Python that allows to register and audio accessing the microphone of my laptop. The duration and the sampling frequency were, for this reason, imposed. I decided to use this method in particular to have a suitable sampling frequency Fs (as the task suggested), for having a good quality analysis. The duration was put equal to 10 seconds, while the sampling frequency to 44100 Hz, i.e. 44.1 kHz. Moreover, the speech was captured in mono format (1 channel) and was put into an unidimensional array.
After the registration I also used the function *wavfile* from the library *scipy* to save the signal in a WAV file.
Below it is showed the code performed for this starting step:

```python
import sounddevice as sd
import numpy as np
import scipy.io.wavfile as wav

#Recording parameters
duration = 10  #10 seconds
sample_f = 44100  #Sampling frequency: 44.1 kHz

print("Start recording...")
my_speech = sd.rec(int(duration * sample_f), samplerate=sample_f, channels=1)
sd.wait()  #End of registration
my_speech = my_speech.flatten()  #Converting in 1D array

#Saving audio
wav.write("recording_my_speech.wav", sample_f, my_speech)
```

To have a initial idea of how our signal appears, we can visualize it through a Time-Amplitude (i.e. width) graph, shown in Figure 1.

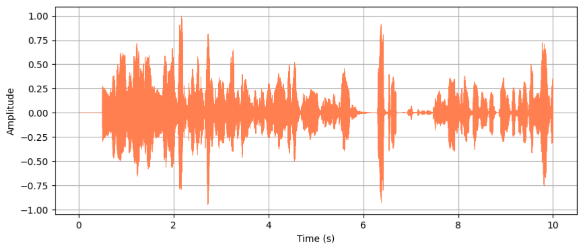


Fig. 1: Speech signal.

3 Basic Speech Signal Characteristics

In this section we calculate some basic speech signal characteristics to have a general overview of it.

- **Duration of the signal (in seconds)**:
  it is the total time of speech's recording. It is defined as:

  $$T_s = \frac{\text{Number of samples}}{\text{Sampling frequency}}$$

  In our case the duration of 10 seconds was placed a priori, but its calculation can still be performed in the following way:

  ```
  duration = len(speech_signal) / fs
  ```

  where fs is the sampling frequency and $len(speech_signal)$ is the number of samples.

- **Sampling frequency and number of samples**:
  they are really connected since the number of samples is defined by the sampling frequency (Fs or fs). The Fs is really important also because it defines the highest frequencies that can be analyzed. By the Nyquist-Shannon Theorem, in fact, in the process of sampling where an analog signal (continuous) is transformed into a digital one (discrete), the sampling frequency has to respect a constraint such that:

  $$f_s \geq 2f_{max}$$

  where f_{max} is the maximum frequency of a signal. Without this, there can be the aliasing phenomenon, where different frequencies overlap and high frequencies can appear as lower ones. Hence, by this theorem, we have that the f_{max}, also called Nyquist frequency, is :

  $$\text{Nyquist frequency} = f_s/2.$$

  Moreover, the number of samples is directly proportional to the duration of the signal and the sampling frequency, so a signal that leasts longer or with higher fs, has a higher number of samples. In our case the fs was set a priori at 44.1 kHz, while the number of samples resulted to be of 441000 samples, since by its formula (seed the point before) it is the product between the duration and the fs.
  The code to obtain them is the following:

  ```
  fs, speech_signal = wav.read('C:/Users/sofyc/OneDrive/Desktop/UPEC/Data capture and processing/
      assignment 2/recording_my_speech.wav')

  #Number of samples
  samples = len(speech_signal)
  ```

  where $len(speech_signal)$ measures basically the length of the signal (i.e. number of samples).

- **Mean value (average amplitude) and variance**:
  it is the arithmetic mean of the values of a signal of its width within time. Specifically, for speech signals, the mean should be very close to 0, since the values oscillates. When it is not like this,

there is probably some noise.
The variance is instead the variability of the speech and measures the spread of signal's values with respect to the mean. The more the variance, the more the width.
The code to calculate them is the following:

```
mean = np.mean(speech_signal)
variance = np.var(speech_signal)
```

The output was a mean of 0.000015 and a variance of 0.0189. As we can see, the mean is very very close to 0 and also the variance is not high. So we have a good quality signal.

- **Energy of the signal (sum of the squared values of the samples)**:
  the energy, as in assignment 1 of ECG signals, is the squared sum of samples' widths within time. It is a measure of the signal's intensity and the larger the width, the higher the energy. In addiction, for speech signals, the energy is really connected to the listening volume. If we have in fact a high energy, the speech will appear us to be stronger.
  The code used to obtain it is the following:

```
energy = np.sum(speech_signal**2)
```

The outputs a energy of 8333.03.

4 Frequency Representation and Spectrum analysis

In this section we analyze the speech signal in the frequency domain, in particular through the Power Spectral Density (PSD), expressed as a function of the frequency, and the dominant frequency. Moreover, we will analyze the spectral components of the signal, passing from the time dimension to the frequency one. In addiction, we will also explore different sampling frequencies and different windowing functions and lengths, evaluating their impact on frequency resolution.

4.1 Power Spectral Density (PSD) and its representation

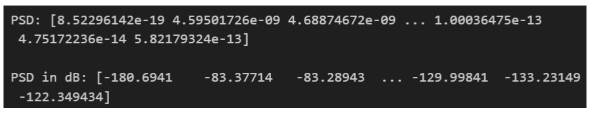
The Power Spectral Density (PSD) provides a measure of how the power distribution of a signal is distributed among frequency. It is defined as:

$$PSD(f) = \lim_{T\to\infty} \frac{1}{T} |\hat{x}_T(f)|^2$$

where $\hat{x}(f)$ is the Fourier transform of the signal $x(t)$, and $|\hat{x}(f)|^2$ is the energy of the signal.
We can display it through the **periodogram**, a method to perform the estimation of the spectral density of the signal. We can perform it through the function *periodogram* from *scipy.signal* module. It is an instrument with which it is possible to catch the frequencies and the PSD of the signal.

```
#Periodogram
frequencies , psd = periodogram(speech_signal , fs)
```

The output of this is composed by two arrays (one for the frequencies and the other for the PSD), and it is shown in Figure 2.



```
PSD: [8.52296142e-19 4.59501726e-09 4.68874672e-09 ... 1.00036475e-13
 4.75172236e-14 5.82179324e-13]

PSD in dB: [-180.6941    -83.37714   -83.28943  ... -129.99841  -133.23149
 -122.349434]
```

Fig. 2: PSD output.

We can also visualize the periodogram through a graph that shows the PSD as a function of the frequency. Usually, low frequencies under 1000 Hz, corresponds to basic components when we speak, as for example the **pitch** of a person (i.e. the fundamental frequency) that is based on our tone voice.

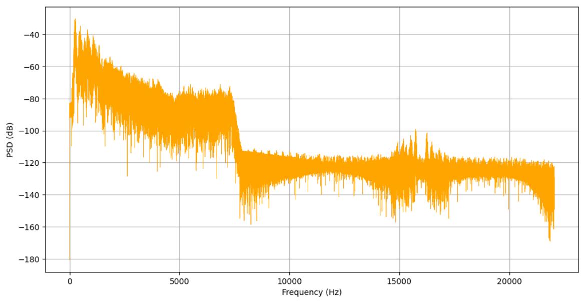


Fig. 3: Power Spectral Density (PSD) of the speech (default window).

For male voices it is usually between 85 Hz and 180 Hz; for female voices between 165 Hz and 255 Hz and for children's voices between 250 Hz to 400 Hz or more.
Regarding high frequencies we can say that they usually represent **formants**. They are basically ”resonances” in our vocal cavity. They are really important for sound quality since they help to identify the phonemes, the principal sounds in a language which can change the meaning of a word (for example in the Italian language /p/ and /b/ are distinct phonemes).
The graph of the periodogram was computed transforming the PSD from Watts$^2/Hz$ to decibel (dB), and it is shown in Figure 3. The conversion was made through the following formula:

$$PSD[dB] = 10log_{10}(PSD\ [\text{Watts}^2/Hz]\ )$$

where PSD[Watts$^2/Hz$] is the Power Spectral Density caught by the function *periodogram*. Also, the window of default in the function for periodogram is the 'boxcar' and it is maintained.
As we can see from the graph, there are peaks in particular in the first few thousand Hz, that could be associated with formants. We can also notice that the power gradually decreases reflecting higher energy in the lower frequencies. We have higher energy with small frequencies' values insce the energy in human vocal signals is concentrated around the pitch, that is not high as we said before.

Furthermore, the frequency resolution of the periodogram is also affected by the duration of the signal: a longer duration allows a better analysis on low frequencies since spectral resolution is inversely proportional to signal length, but at the same time it can leads to computational complexity and also memory stress. In this sense short signals don't provide an accurate representation of low frequencies.

4.2 Windowing functions

The usage of windowing functions such as Hamming and Hanning, is important to reduce the undesirable effects of aliasing (mentioned in a previous section), and spectral “leakage,” that occurs when the energy of a signal spreads in other frequencies causing distortions.
Going more deeply, we have that the **Hamming window** reduces this effect of “leakage”, maintaining a good spectral resolution.
Hanning window, instead, provides a stronger attenuation in the components out of the principal band.
The command to perform this analysis is the following one:

```python
#Periodogram with Hamming window
freq_hamming, psd_hamming = periodogram(speech_signal, fs, window='hamming')

#Periodogram with Hanning window
freq_hanning, psd_hanning = periodogram(speech_signal, fs, window='hann')

#Conversions in decibel (dB)
psd_hamming_db = 10 * np.log10(psd_hamming)
psd_hanning_db = 10 * np.log10(psd_hanning)
```

To have a better idea of the differences between the outputs, we can also visualize the periodograms with different windows. Looking at the Figures 4 and 5 with the periodograms of the two described

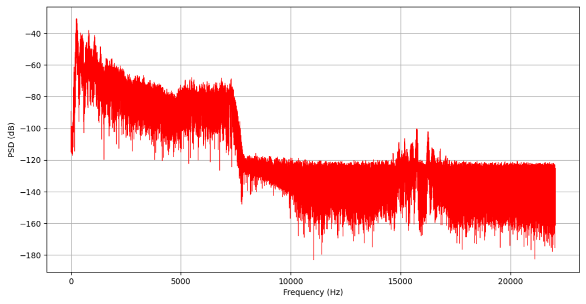


Fig. 4: Power Spectral Density (PSD) of the speech with Hamming window

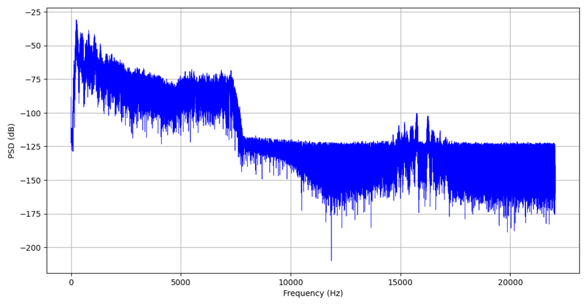


Fig. 5: Power Spectral Density (PSD) of the speech with Hanning window.

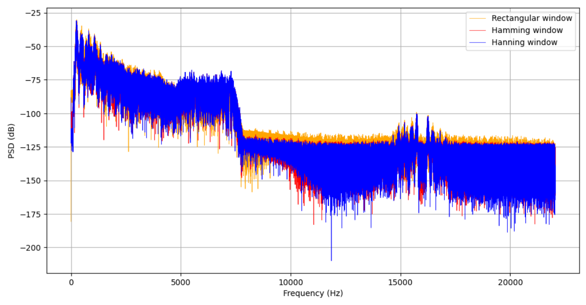


Fig. 6: Power Spectral Density (PSD) of the speech.

windows, we can see that there is a gradual amplitude attenuation in high-frequency components, while with Hanning window, a strong reduction is evident in the frequencies away from the main peaks and the graph has fewer high-frequency oscillations but slightly lower spectral resolution than the Hamming window.

Finally, to have another precise idea of the differences and make an overall comparison, Figure 6 shows PSD (dB) -Frequency (Hz) in the three cases: in orange for the default window boxcar, in red for Hamming window and in blue for Hanning window. We can notice that the Hamming window outputs a "smoother" spectral representation with less distortion in the high-frequency peaks. In contrast, the Hanning one smoothes oscillations out of the band more effectively, but lose some detail on secondary peaks.

4.3 Dominant frequencies

Dominant frequencies represent principal peaks in spectral analysis and indicates frequencies where the signal has maximum energy. They were caught with these commands:

```
#Dominant frequencies
three_dom_idx = np.argsort(psd)[-3:][::-1]
```

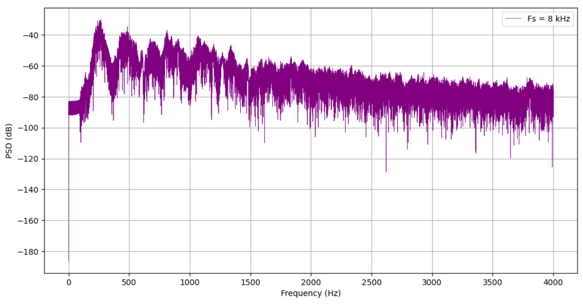


Fig. 7: Speech recognition with Sampling Frequency of 8 kHz.

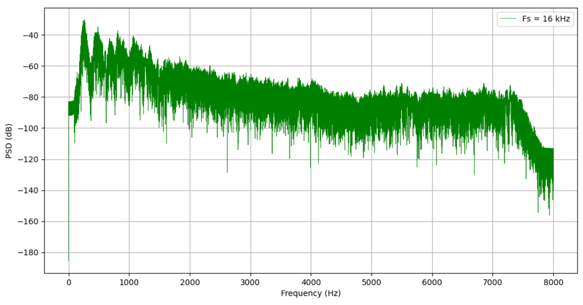


Fig. 8: Speech recognition with Sampling Frequency of 16 kHz.

```
freq_dominant_3 = frequencies[three_dom_idx]
```

The outputs were the following:

- First dominant frequency: 265.90 Hz
- Second dominant frequency: 265.80 Hz
- Third dominant frequency: 256.30 Hz

Since they are close to each other, they suggest that there are harmonics in the signals, specific for human voice (they are close to the pitch).

4.4 Different sampling frequencies

This section of the code shows how the speech signal is resampled at different frequencies, the ones suggested in the task: 8 kHz and 16 kHz. Resampling is useful to compare how the change in sampling frequency affects the spectral representation. Also, after resampling, the PSD and corresponding frequency are calculated again, allowing analysis of whether there are significant differences in the spectral representation of the signal at different sampling frequencies.
The commands are the following:

```
#Resample at 8 kHz and 16 kHz
my_speech_8k = resample(speech_signal, int(len(speech_signal) * 8000 / fs))
my_speech_16k = resample(speech_signal, int(len(speech_signal) * 16000 / fs))

#PSD for the new fs
freq_8k, psd_8k = periodogram(my_speech_8k, 8000)
freq_16k, psd_16k = periodogram(my_speech_16k, 16000)

#Conversions in decibel (dB)
psd_8k_db = 10 * np.log10(psd_8k)
psd_16k_db = 10 * np.log10(psd_16k)
```

The resulting graphs are shown in Figures 7 and 8, and as we can see as the sampling frequency is reduced, the x axis of the periodogram has a lower range.

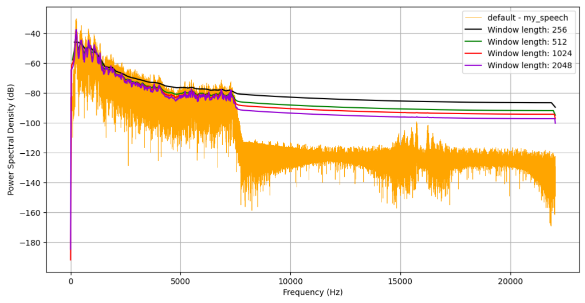


Fig. 9: Impact of Varying Window Lengths on Frequency Resolution.

4.5 Different window lengths

In this final section, we explore how window lengths affect the frequency resolution of spectral analysis. In fact, different window lengths can alter the precision with which frequencies appear in the period. We used windows of different lengths (256, 512, 1024, and 2048 samples), to examine how the choice of window length affects PSD and spectral detail. As we can see from Figure 9, longer windows provides better frequency resolution, while a shorter window can better capture temporal variations in the signal.
The commands are the following:

```python
#Array with different window lengths on the frequency resolution
window_lengths = [256, 512, 1024, 2048]
windows = [get_window('boxcar', length) for length in window_lengths] #Default window
colors = ['black', 'green', 'red', 'darkviolet']
```

References

[1] Scicoding: Calculating Power Spectral Density in Python. https://www.scicoding.com/calculating-power-spectral-density-in-python/. Accessed: 2024-10-11

[2] Developers, S.: SciPy Signal Documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.periodogram.html. Accessed: 2024-10-11

[3] Developers, M.: Source Power Spectrum. https://mne.tools/1.7/auto_examples/time_frequency/source_power_spectrum.html. Accessed: 2024-10-11

[4] Contributors, W.: Spectral Density. https://en.wikipedia.org/wiki/Spectral_density. Accessed: 2024-10-11

[5] Nait-Ali, A.: Session: Spectral Analysis. Lecture slides, Université Paris-Est Crétéil (UPEC) - Faculté des Sciences et Technologie (2024)

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