First Homework Assignment

of Sound Analysis, Synthesis and Processing

Anna Fusari
Filippo Marri
8 maggio 2024

General Description:

The aim of this homework is to perform DOA estimation. Simply speaking, we want to identify the angular position with respect to the midpoint of an array of 16 microphones for a series of time instants by means of delay-and-sum beamformer technique.

Mathematical Explanation:

Mathematically speaking, in order to solve the problem we have to perform spatial filtering for every frequency and every instant of time by projecting the signal along different basis that are, in turn, determined by different spatial frequencies ω_s for the specific frequency ω_c we are evaluating. After that, we can evaluate a function $p(t,\theta)$ whose maximum points related to the variable θ for each value of t indicate the angular position of the source.

Therefore, fixed a certain frequency and a certain instant of time, our first step is to design a filter $\underline{h}\left(\overline{\theta}\right)$ that could perform spatial frequency decomposition along the angle $\overline{\theta}$. We design our filter as an optimisation problem with a constrain that is: the sound coming from the direction $\overline{\theta}$ has to pass undistorted, while the sound coming from other directions has to be attenuated. This is just the corresponding of the band pass filter in the spatial domain.

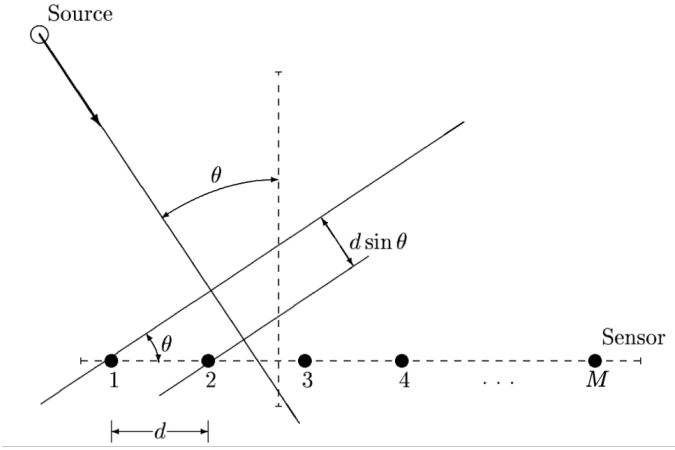

Figure 1. Representation of the ULA and of the trigonometrical decomposition of the delay.

The condition “let the signal coming from $\overline{\theta}$ untouched” is formalised by the expression:

$$\underline{h}^H\left(\overline{\theta}\right)\underline{a}\left(\overline{\theta}\right)=1$$

Where $\underline{a}\left(\overline{\theta}\right)$ is the so-called propagation vector related to the angle $\overline{\theta}$ defined by:

$$\underline{a}\left(\overline{\theta}\right)=\begin{pmatrix}H_1\left(\omega_c\right)e^{-j\omega_c\overline{\tau}_1}\\H_2\left(\omega_c\right)e^{-j\omega_c\overline{\tau}_2}\\\vdots\\H_M\left(\omega_c\right)e^{-j\omega_c\overline{\tau}_M}\end{pmatrix}\underset{\overline{\tau}_k=(k-1)\frac{d\sin(\overline{\theta})}{c}}{=}\begin{pmatrix}H_1\left(\omega_c\right)e^{-j\omega_c\cdot 0}\\H_2\left(\omega_c\right)e^{-j\omega_c(1)\frac{d\sin(\overline{\theta})}{c}}\\\vdots\\H_M\left(\omega_c\right)e^{-j\omega_c(M-1)\frac{d\sin(\overline{\theta})}{c}}\end{pmatrix}\underset{\overline{\omega}_s\overset{def}{=}\omega_c\frac{d\sin(\theta)}{c}}{=}\begin{pmatrix}H_1\left(\omega_c\right)e^{-j\omega_c\cdot 0}\\H_2\left(\omega_c\right)e^{-j(1)\omega_s}\\\vdots\\H_M\left(\omega_c\right)e^{-j(M-1)\omega_s}\end{pmatrix}$$

In our particular condition, the parameters are:

- $M=16$: number of omnidirectional and identical $\left(H_k\left(\omega_c\right)=1\ \forall k\right)$ microphones;
- $d=\frac{L}{M}=\frac{40\ cm}{16}=2.5\ cm$: distance between microphones;
- $c=343\ \frac{m}{s}$: speed of sound.

What we get is:

$$\underline{a}\left(\overline{\theta}\right)=\begin{pmatrix}1\\e^{-j\omega_c\frac{0.025\cdot\sin(\overline{\theta})}{343\frac{1}{s}}}\\\vdots\\e^{-j15\omega_c\frac{0.025\cdot\sin(\overline{\theta})}{343\frac{1}{s}}}\end{pmatrix}$$

If we inspect the propagation vector, we can observe that basically contains the samples of a complex sinusoid. Therefore, what the microphone performs is nothing but a spatial sampling of the sound field. Like any other sampling operation, it has to satisfy the Nyqvist's sampling condition

$$|\omega_s| \leq \pi$$

From which we derive a condition on the maximum value of frequency $f_c = \frac{\omega_c}{2\pi}$ that we can use.

$$f_c \leq \frac{c}{2d} = \frac{343\ \frac{m}{s}}{2 \cdot 0.025\ m} = 6860\ Hz$$

We were saying to derive the weights of the filter by solving an optimisation problem that can be formalised as:

$$\underline{h}\left(\overline{\theta}\right) = \underset{\underline{h}}{\operatorname{argmin}} \left(\underline{h}^H \underline{h}\right) \qquad subject\ to\ \underline{h}^H\left(\overline{\theta}\right)\underline{a}\left(\overline{\theta}\right) = 1$$

This problem has a close form solution:

$$\underline{h}\left(\overline{\theta}\right) = \frac{\underline{a}\left(\overline{\theta}\right)}{M} = \frac{\underline{a}\left(\overline{\theta}\right)}{16}$$

When we change the direction $\overline{\theta}$ (and consequently the propagation vector), we get, since the propagation vector is basically a set of delays (complex exponentials), a sum of the delayed samples of the microphone. The delay is determined by the propagation vector $\underline{a}\left(\overline{\theta}\right)$ that, when it does

not match the actual DOA, will build disruptive summation minimising the energy. That is why the filter is called delay-and-sum beamformer: we delay the signal using the propagation vector $\underline{a}\left(\overline{\theta}\right)$ and we sum the output of the delayed signal. As we just said, if we match the actual propagation vector the sum will be constructive and the energy maximised. On the other hand, disruptive.

If we plug this definition inside the power of the filtered output $y_F(t) = \underline{h}^H \underline{y}(t)$, we obtain the following formula:

$$E\{|y_F(t)|^2\} \underbrace{=}_{y_F(t)=\underline{h}^H(\overline{\theta})\underline{y}(t)} E\{|\underline{h}^H\left(\overline{\theta}\right)\underline{y}(t)|^2\} = E\{\underline{h}^H\left(\overline{\theta}\right)\underline{y}(t)\underline{y}^H(t)\underline{h}\left(\overline{\theta}\right)\} \underbrace{=}_{\underline{h}(\overline{\theta}) \text{ is deterministic}}$$

$$\underline{h}^H\left(\overline{\theta}\right)E\{\underline{y}(t)\underline{y}^H(t)\}\underline{h}\left(\overline{\theta}\right) \underbrace{=}_{\underline{\underline{R}}=E\{\underline{y}(t)\underline{y}^H(t)\}} \underline{h}^H\left(\overline{\theta}\right)\underline{\underline{R}}\underline{h}\left(\overline{\theta}\right) \underbrace{=}_{\underline{h}(\overline{\theta})=\frac{\underline{a}(\overline{\theta})}{16}} \frac{\underline{a}^H\left(\overline{\theta}\right)}{16}\underline{\underline{R}}\frac{\underline{a}\left(\overline{\theta}\right)}{16} =$$

$$= \frac{\underline{a}^H\left(\overline{\theta}\right)\underline{\underline{R}}\underline{a}\left(\overline{\theta}\right)}{16^2} \underbrace{\simeq}_{\underline{\underline{R}}=E\{\underline{y}(t)\underline{y}^H(t)\}\simeq\frac{1}{K}\sum_{t=1}^{K}\underline{y}(t)\underline{y}^H(t)=\hat{\underline{\underline{R}}}} \frac{\underline{a}^H\left(\overline{\theta}\right)\hat{\underline{\underline{R}}}\underline{a}\left(\overline{\theta}\right)}{16^2}$$

We conclude that in order to estimate the Direction of Arrival of the sources, we just have to evaluate the following function at all the spatial directions θ:

$$p\left(\theta\right) = \frac{\underline{a}^H\left(\theta\right)\hat{\underline{\underline{R}}}\underline{a}\left(\theta\right)}{16^2} \qquad with\ \theta \in \left[-90°, 90°\right]$$

In our case, this function is evaluated for every instants of time and, in turns, the function evaluated for every instant of time is evaluated for every frequency. This last function is averaged along the frequency dimension so that we obtain a three-dimensional function defined over the bi-dimensional domain (t, θ).

Code analysis:

We are ready now to implement the algorithm we have just explained. In order to better organise the code, we decided to arrange it in different functions:

ourStft:

The function “ourStft” compute the STFT of a multi-channel signal. The arguments are:

- `s`: the input signal.
- `sampling_frequency`: nomen omen, the sampling frequency.
- `window_size`: the length of the window use to crop the signal.
- `hop_size`: the length of the portion of samples that do not overlap between different windows.

First we want to initialise an hamming window, because it is the best choice since its shape helps to crop the signal in a smooth way.

Then we set two variables that we will use in the function:

- `N`: the length of `s` expressed in samples.
- `M`: the number of windows; we evaluate that by subtracting from the `N` samples the length of one window and dividing that number for the `hop_size`.
- `S`: the initialisation of the tensor that will contain the STFT of the signal, the first dimension corresponds to the frequency, so the `window_size`, the second represents the time, so the number of frames (windows, `M`) and the last one depicts the channel.

We are now ready to get into the thick of the STFT implementation by setting the first cycle that selects one channel at a time. In order to perform the Fourier transform we cannot work on a tensor of three dimensions but we

need to take just the information about time and frequency. So, we initialise the matrix `s_plane` in which we store the information for every channel.

After that, we start cycling over the number of windows, and we use `s_plane` to create the cropped version of the signal `s_cropped`.
In order to perform the segmentation of `s_plane` we consider the frame of length `window_lenght` relative to the index of the current window, and we apply to it the hamming window using the element wise multiplication.

It is now the time to evaluate the Fourier transform of the `s_cropped` signal with the built-in function `fft`. The result is stored into the `S_cropped` matrix.

The last step is to centre the STFT in zero. To do that, we use three different variables:

- `k`: half the number of windows.
- `X`: matrix that saved the first `k` time instant from `S_cropped`.
- `Y`: matrix that saved the flipped version of `X`, so the negative side of the STFT.

Now we store `Y` inside the first `k` time instants of `S`, this is the negative part of the STFT, after that we save the positive side `X` starting from the index `k+1`. The figure 3 shows the STFT centred in zero and the comparison with the version made by the built-in Matlab function `stft`.

In the end, we set two important vectors, `time_axis` and `frequency_axis`. They represent, respectively, the vector containing the time instants and the one containing the frequencies. The function return these two with the STFT `S`.

threeDPlot:

The threeDPlot is used in the code to plot a 3D version of the module of the STFT signal.

The arguments of the function are:

- `S` that represents the STFT of the signal to plot.
- `f`: the frequency axis.
- `t`: the time axis.

In order to obtain a plot of the `S` function as a third dimension above a grid in the (x, y) plane, we decide to use the built-in `waterfall` function.

powerEstimation:

The function `powerEstimation` returns the pseudo-spectrum evaluated by an array of microphone of a sound field, given as input:

- `S`: a tensor containing the STFT (frequency on the first dimension and time on the second one) for every channel (third dimension).
- `f`: a vector that contains the frequency bins of the STFT.
- `t`: a vector that contains the time instants of the STFT.
- `Fs`: the sampling frequency.
- `numberOfMicrophones`: a variable containing the number of microphones that compose the array.
- `meanComp`: a parameter that allows us to choose which kind of mean to compute. We will see later in which context.
- `normComp`: a parameter that allows us to choose whether or not to normalise the pseudo-spectrum over time.

First of all, since we are dealing with real signals, we know that for their fft holds the hermitian symmetry. In light of this, to optimise the algorithm we can evaluate just the positive frequencies of the spectrum and neglect the

negative ones whose respective STFT values are nothing but the negative complex conjugate counterparts of the positive ones.

After that we initialise the `pseudoSpectrum` matrix whose rows will represent the time instants while the columns the angles.

We are ready now to perform the actual pseudo-spectrum estimation. We get into a for loop that cycles over the number of angles. For each angle, we assign to the variable `theta` the value of the angle related to the index of the for loop we are considering. At this point we evaluate the array of `delays` $\bar{\tau}_k = \left(k-1\right)\frac{d\sin\left(\bar{\theta}\right)}{c}$ for all the microphones (so for $k \in \left[1;16\right]$) where d is the `distanceBetweenMic` and c the `soundSpeed`. After that, we get into a nested for loop over the time instants. In there, we initialise the array `beamformedSignal` as a row vector of length `numberOfMicrophones`. We have now to fill this vector, and we do that with another nested for loop in which we cycle over the `numberOfMicrophones`. In here, we save in a variable called `micSignal`, the STFT value sampled at the frequency bin and time instant identified by the counter variable of their relative for loops for the microphone we are considering. Then, we fill the `beamformedSignal` vector by putting inside it the projection of the `micSignal` on the base $e^{-j2\pi\frac{f_c}{F_s}\bar{\tau}_k}$. Out of this for loop we compute the modulus squared of the `beamformedSignal` vector, that we save in a value called `power`. As we can notice, we preferred to split the power-spectrum computation in different steps by solving the problem with the most linear approach. In this way we are able to highlight and explain every single operation. Obviously, it is not the fastest way: it is surely better to substitute the nested for loops with matricial products as we explained in the “mathematical explanation” paragraph.

Before going out from the last for loop, we store the `power` value in a tensor called `powerPreAvereged` that will contains the power for each frequency index.

Now we perform the average along the frequency axis according to the parameter we set. There are three different kinds of mean that we can perform:

- Geometric mean: a mean which indicates a central tendency of a finite set of real numbers by using the product of their values

$$\texttt{geoMean} = \left(\prod_{i=1}^{N} x_i \right)^{\frac{1}{N}}$$

- Harmonic mean: it is expressed as the reciprocal of the arithmetic mean of the reciprocals of the number of elements in the given set of observations

$$\texttt{harmMean} = \frac{N}{\sum_{i=1}^{N} \frac{1}{x_i}}$$

- Arithmetic mean: it can be evaluated as the sum of all of the values divided by the number of values.

$$\texttt{mean} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

At the end, if the `normComp` parameter is set to `true`, we normalise all the values of the `pseudoSpectrum` along the time axis.

main:

In the main file we simply put things together. After cleaning our workspace, we import the audio file *array_recordings.wav* in a variable called `audioData` and we save the sampling frequency $F_s = 8\ KHz$ in the variable `Fs`. In this way, `audioData` will be a matrix whose rows represent the time intents and whose columns represent the microphones. Each cell will contain

the value of the signal sampled at the instant of time identified by the row for that specific microphone. Before going on, let us dwell on the highest frequency we will consider in our analysis: our STFT performs the frequency decomposition up to the Nyqvist frequency, thus up to $\frac{F_s}{2} = 4\ kHz$. This frequency is lower than the maximum value of frequency that we can analyse with this model $f_c = 6860\ Hz$. Therefore, we can proceed without cutting off any frequency of our signal.

At this point, we set some parameters that will come in handy in the further implementation. We highlight that we decided not to define all the parameters at one time at the beginning of the code, but inside every code-section in order to group the elements related to a same topic together.
For what it concerns the definition of the problem and the audio load, the parameters are:

- `L`: length of the microphone array ($L = 0.4\ m$)
- `numberOfMicrophones`: nomen omen, the number of microphones that compose the array (16) and, consequently, number of channels of the audio file.
- `distanceBetweenMicrophones`: it is evaluated as the ratio between the length of the array `L` and the number of microphones. We highlight that we subtract one since we start counting the microphones from 0.
- `N`: it contains the length of the audioData file expressed in samples.
- `time_axis`: it is a vector that contains the values of time instants expressed in seconds.

In order to see if we loaded all the audio files correctly, we plot the sixteen audio-waves.

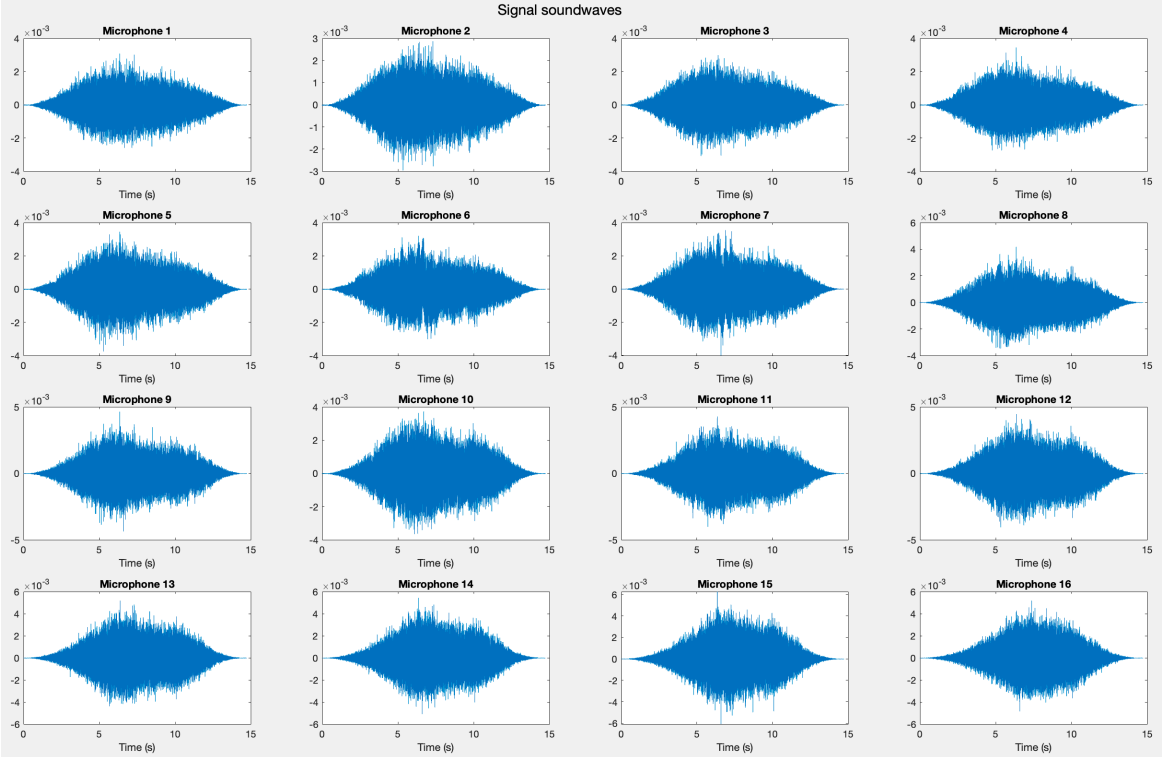

Figure 2. Signal sound waves

We are now ready to perform the proper algorithm. Since the delay-and-sum beamforming technique is a narrow band one, we need to find an expedient to deal with our wide-band signal. A simple idea is to decompose the signal in the series of its frequency components for every instant of time and to perform the technique for a single frequency. At the end we will average the result along frequencies. The decomposition we have just described is nothing but the STFT. Once again, we define some useful parameters:

- `windowLength`: length of the window used to crop the signal in order to perform, inhere, the fft. We set it equal to 1024 samples. We think that

this number is a good compromise to obtain a satisfying result for what it concerns both the frequency and the time resolutions.

- `hopSize`: so the step size in which the window of the STFT is to be shifted across the signal. Simply speaking, how much we can advance the analysis time origin from frame to frame. The value is 512, therefore an `hopSize` of 50% of the window.

At this point we call two functions to perform the STFT: the former is our STFT implementation, the latter the MATLAB's built-in one. We decided to plot both the functions to compare the two results and understand how close our outcome was with the MATLAB's one. Obviously, we used the same parameter for the two functions. We plot two modules for one channel by using the `threeDPlot` function.

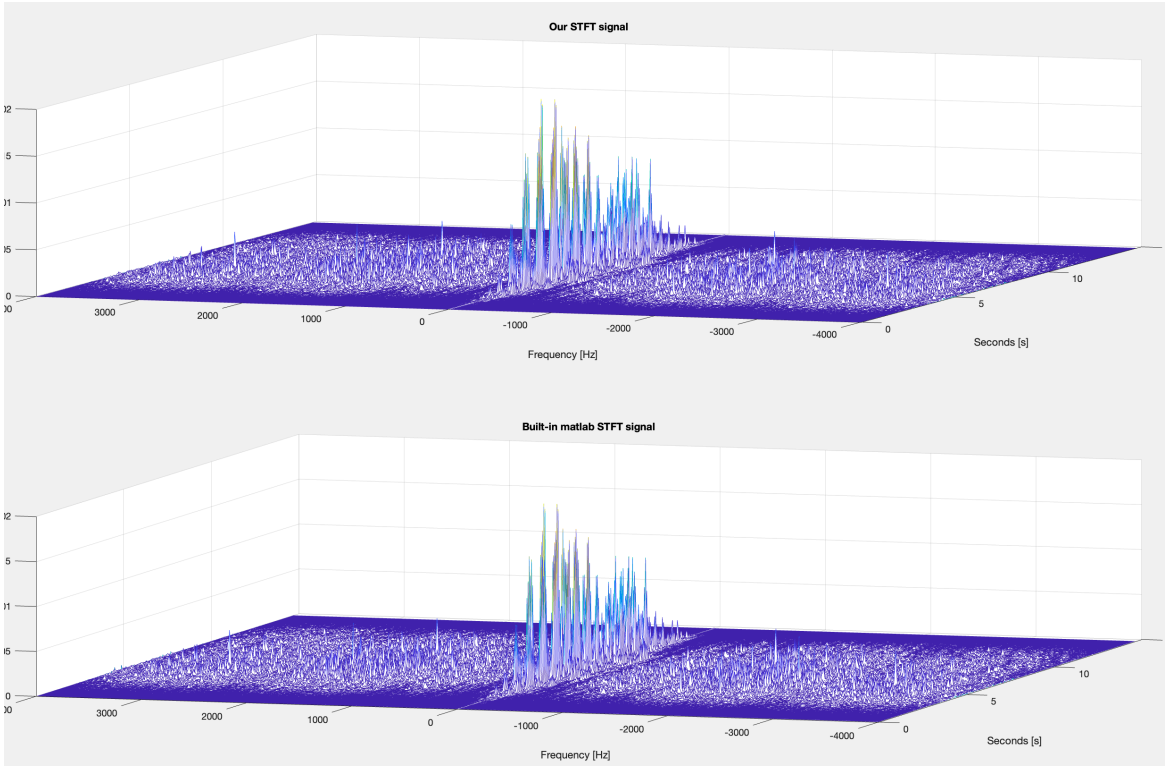

Figure 3. STFT comparison

Now that we saved the STFT in the tensor `S` (whose first dimension represents the frequency bins, whose second dimension represents the time instants and whose third dimension represents the channels), we can perform the proper computation of the pseudo-spectrum. Times for defining new parameters:

- `soundSpeed` in air, approximated to $343\ \frac{m}{s}$
- `azimuthAngle` is a vector containing all the integer angular values between -90° and 90°.

We call now the `powerEstimation` function that takes as input the tensor `S`, the frequency vector `f` containing all the values of frequency, the vector `t` containing the time instants, the sampling frequency `Fs`, the `numberOfMicrophones`, the `azimuthAngle` vector, `distanceBetweenMic`, the `soundSpeed` and two parameters that we set respectively `"geometric"`, in order to select the geometric mean in the function, and `true` to normalise the pseudo-spectrum. We save the result of the function in the variable `pseudoSpectrum`. Then, we retrieve the point of maxima of the pseudo-spectrum along the time axis: this means that the function returns a vector containing all the indexes of the angle for which the pseudo-spectrum shows a maximum for every time instant. We save this result in the vector `maxIndex`. We convert the indexes into the actual angle values by simply picking them from the `azimuthAngle` vector and storing them into another vector called `maxAngle`.

In order to see if we achieved the right result, we plot both the pseudo-spectrum and the maxAngle vector.

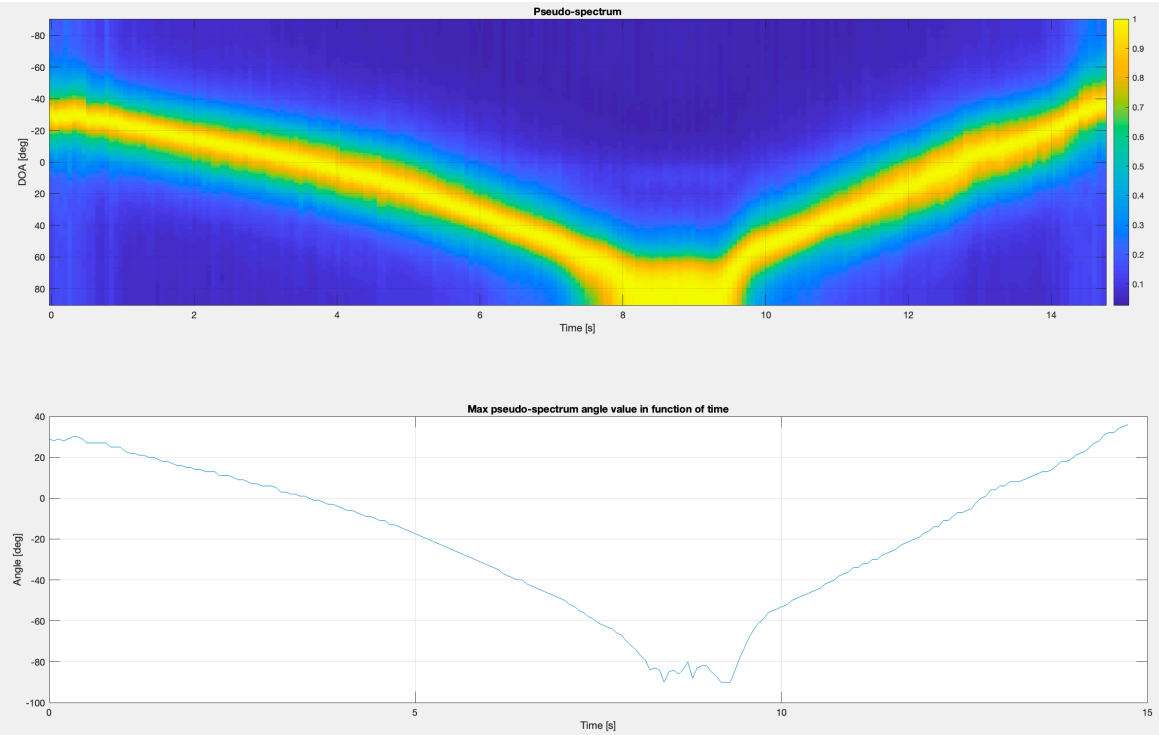

Figure 4. Pseudospectrum representations.

We notice that the second plot recalls exactly the maximum trajectory presents in the first one. Exactly what we expected to get.

As last step, we create a video that shows the evolution of the DOA, represented by an arrow. To perform this task we use the function `VideoWriter` and, then, we open the video file we created. We open a figure, where we represent the 16 microphones with a series of dots. Then, by means of a for cycle, we plot the position of the source (identified by an arrow that represents the relative DOA) using the `quiver` function that, in turn, basically draws an arrow starting from the (x,y) coordinates given as the first two arguments and ending in the point identified by the couple `(x_tip, y_tip)` given, respectively, as third and fourth argument. The for cycle allows us to create a series of photograms saved as frames by the `getframe`

function. We save the series of frames in our video by means of the `writeVideo` function and, finally, we close the video file.

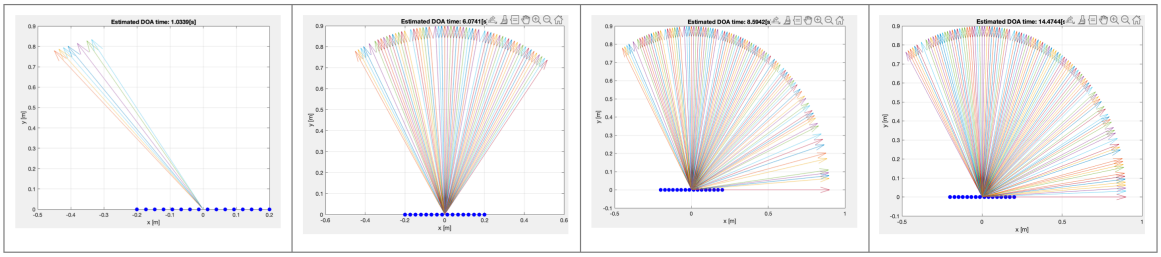

Figure 5. Four frames of the video representing the evolution of the DOA at different time instants

Discussion of the results:

A first discussion can be conducted on the STFT's parameters `windowLength` and `hopSize`. As we increase the `windowLength` we also increase the computational burden of the algorithm that gets slower and slower. However, if we increase the value of the `windowLength` we take into account more samples on which evaluate the STFT. This means that on the one hand we will work on a larger number of frequencies but, on the other hand, that the evaluation of the STFT will be less specific, given that we are substantially reducing the time resolution. It is better working with a large number of frequencies in order to get a more accurate result when we perform the beamforming technique. To compensate the reduction of time resolution we can reduce the value of `hopSize` because, by choosing a smaller one, we evaluate the STFT on a smaller number of samples at a time. We conclude that, it is better choosing relatively high values for `windowLength` and relatively small values for `hopSize`. We have also to pay attention not to be too much extreme: if we choose a value of `windowLength` that is too much high and a value of `hopSize` that is too much low we lose accuracy. It is, as always, a matter of trade-off.

Another parameter that drastically changes the results is the type of mean that we use to average along the frequencies dimension in the beamforming function. We report below the results that we get with different mean functions:

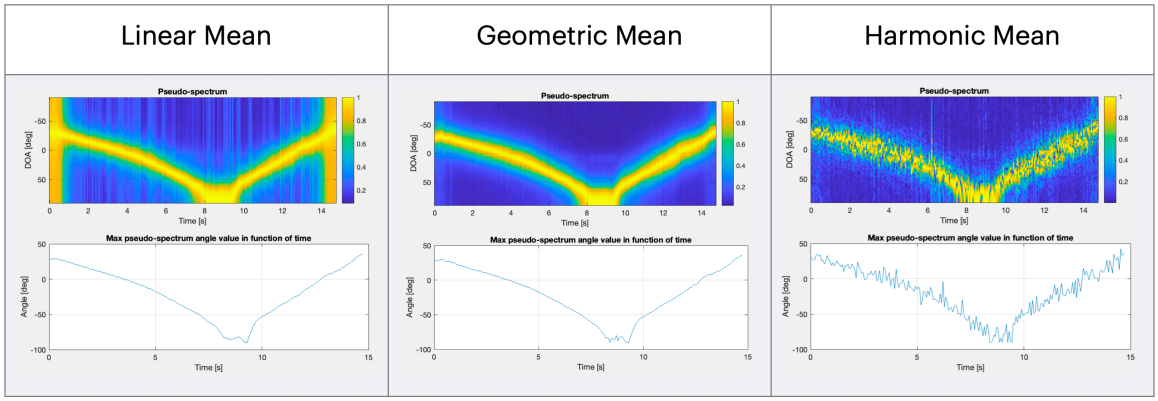

Figure 6. Comparison between the results obtained with different type of means

As we can notice, between the threes, the geometric mean is the one that minimises the error.

The last aspect that we can point out is why to normalise the pseudo -spectrum. This is basically to obtain a clearer distinction between the maximum and the other values. We report a comparison between the non-normalised pseudos-spectrum and the normalised one.

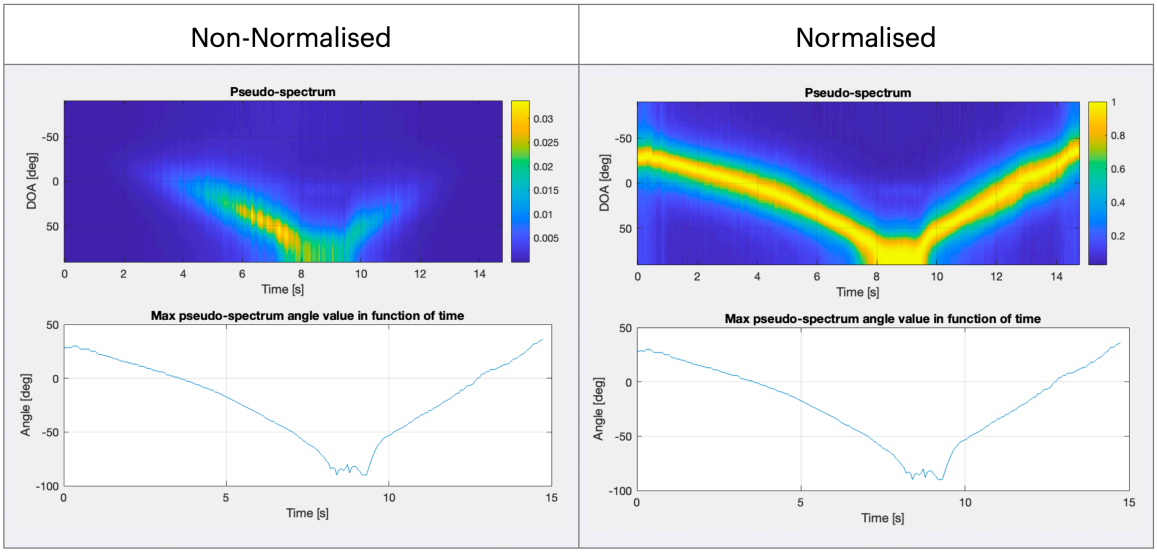

Figure 7. Comparison between the non-normalised and normalised result.

References:

1. MathWorks, *audioread*, https://it.mathworks.com/help/matlab/ref/audioread.html
2. MathWorks, *Short-Time Fourier Transform (STFT) with Matlab*, https://it.mathworks.com/matlabcentral/fileexchange/45197-short-time-fourier-transform-stft-with-matlab
3. MathWorks, *stft*, https://it.mathworks.com/help/signal/ref/stft.html
4. MathWorks, *fft*, https://it.mathworks.com/help/matlab/ref/fft.html
5. MathWorks, *waterfall*, https://it.mathworks.com/help/matlab/ref/waterfall.html
6. MathWokrs, *hamming*, https://it.mathworks.com/help/signal/ref/hamming.html
7. MathWokrs, *geomean*, https://it.mathworks.com/help/stats/geomean.html
8. MathWorks, *harmmean*, https://it.mathworks.com/help/stats/harmmean.html#
9. MathWorks, *mean*, https://it.mathworks.com/help/matlab/ref/mean.html#
10. MathWorks, *VideoWriter*, https://it.mathworks.com/help/matlab/ref/videowriter.html
11. YouTube, *Making Videos From Figures In Matlab*, https://www.youtube.com/watch?v=mvXJh_TDKG8&t=2s
12. MathWorks, *Quiver*, https://it.mathworks.com/help/matlab/ref/quiver.html