Angelo Antona, Alessandro Manattini

Sound Analysis, Synthesis, and Processing  Module 1

SASP HOMEWORK REPORT

Contents

1 Folders Map ......................................................................................................................................... 3

[2 Project Overview 3](#_Toc165990670)

[3 Development and Testing Approach 3](#_Toc165990671)

[4 Component Descriptions and Tests 4](#_Toc165990672)

[4.1 AudioData Class 4](#_Toc165990673)

[4.1.1 Functionality 4](#_Toc165990674)

[4.1.2 AudioData Test in MainTest 4](#_Toc165990675)

[4.2 CustomFFT Function 5](#_Toc165990676)

[4.2.1 Functionality 5](#_Toc165990677)

[4.2.2 Implementation Details 5](#_Toc165990678)

[4.2.3 CustomFFT Test in MainTest 5](#_Toc165990679)

[4.3 STFTProcessor Function 6](#_Toc165990680)

[4.3.1 Functionality 6](#_Toc165990681)

[4.3.2 Implementation Details 6](#_Toc165990682)

[4.3.3 STFTProcessor Test in MainTest 7](#_Toc165990683)

[4.4 AllChannelSTFT Function 7](#_Toc165990684)

[4.4.1 Functionality 7](#_Toc165990685)

[4.4.2 Implementation Details 7](#_Toc165990686)

[4.4.3 AllChannelSTFT Test in MainTest 8](#_Toc165990687)

[4.5 GetCovMatrix Function 8](#_Toc165990688)

[4.5.1 Functionality 8](#_Toc165990689)

[4.5.2 Implementation Details 8](#_Toc165990690)

[4.5.3 GetCovMatrix Test in MainTest 9](#_Toc165990691)

[4.6 GetSteeringVector Function 9](#_Toc165990692)

[4.6.1 Functionality 9](#_Toc165990693)

[4.6.2 Implementation Details 9](#_Toc165990694)

[4.7 Beamform Function 10](#_Toc165990695)

[4.7.1 Functionality 10](#_Toc165990696)

[4.7.2 Implementation Details 10](#_Toc165990697)

[4.7.3 Test in MainTest 11](#_Toc165990698)

[4.8 DOAEstimator Function 11](#_Toc165990699)

[4.8.1 Functionality 11](#_Toc165990700)

[4.8.2 Implementation Details 11](#_Toc165990701)

[4.8.3 Test in MainTest 12](#_Toc165990702)

[4.9 Visualization Functions (VisualizePseudospectrum and VisualizeDOAestimates) 13](#_Toc165990703)

[4.10 Video Generation (getSingleFrame, FramesGenerator, VideoGenerator) 13](#_Toc165990704)

[4.11 Main Script Functionality 13](#_Toc165990705)

[5 Results Of The Provided Signal Analysis 14](#_Toc165990706)

[5.1 Analysis of the Pseudospectrum 14](#_Toc165990707)

[5.2 Analysis of DOA Estimates 15](#_Toc165990708)

[5.3 Conclusion 16](#_Toc165990709)

# Folders Map

* **Audio Files** : it contains the audio of the project.
* **Plots** : it contains the images of the plots.
* **Tests** : it contains the MainTest function with the correlated functions used for the initial testing .
* **VideoFrames** : it contains the video and the frames of the visualization of the single DOA of the geometric mean.
* **VideoFramesMulti** : it contains the frames and the video of the parallel vision of the 3 types of mean.

# Project Overview

A graph of a function

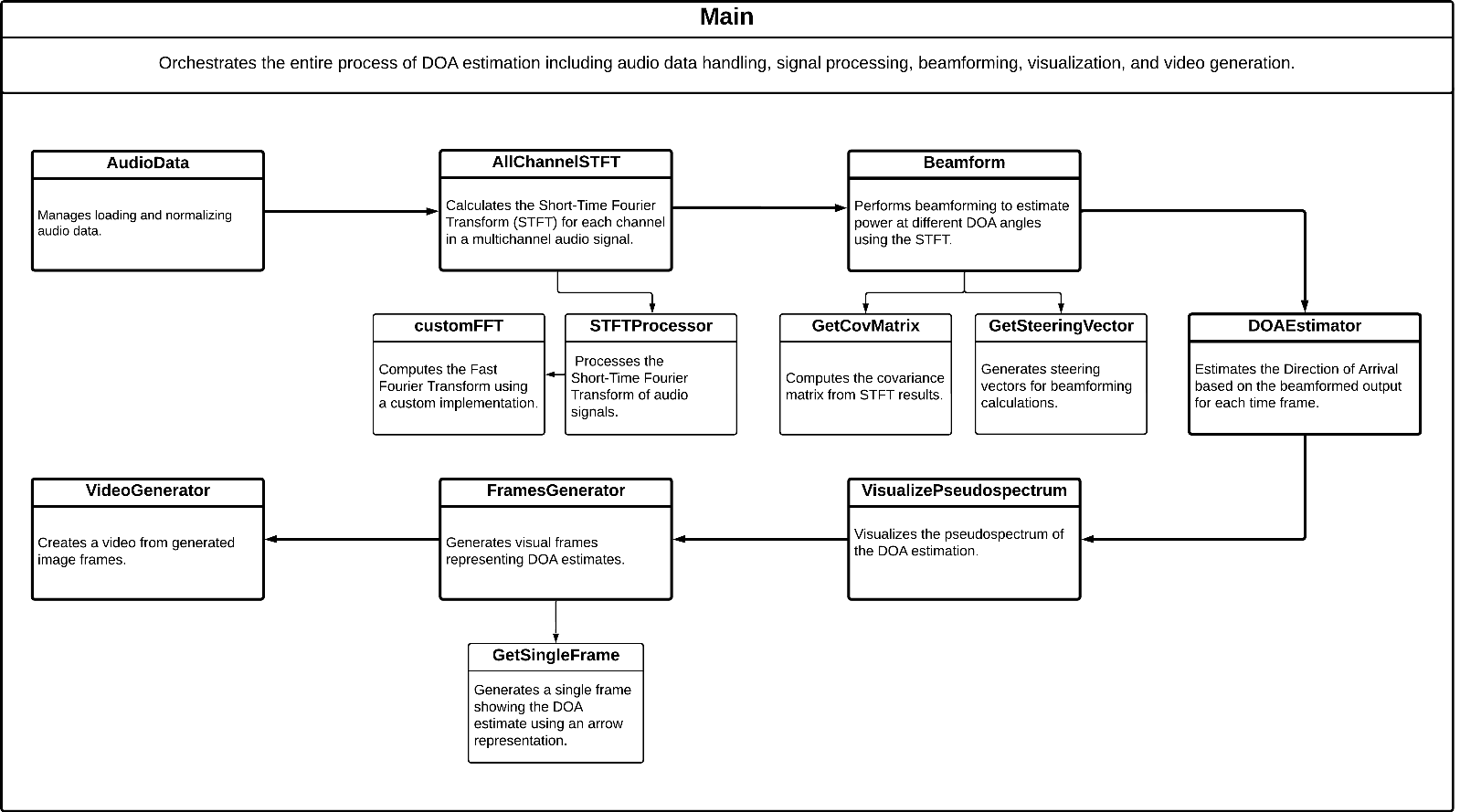
Description automatically generatedThis project aims to implement Direction of Arrival (DOA) estimation using audio signal processing techniques. The source localization involves a multichannel recording acquired using a uniform linear microphone array (ULA) composed of 16 MEMS microphones spaced along a 45 cm length. This array captures audio data via an Audio-over-IP connection at a sampling rate of 8 kHz. The acoustic scene features a moving sound source in front of the ULA, and the speed of sound is established at 343 m/s.

Our objective is to implement a delay-and-sum beamformer tailored for localizing wide-band sources. Given the narrow-band nature of spatial filtering, the project handles each frequency band independently and performs spatial filtering frame by frame to accommodate the source's time-varying nature.

# Development and Testing Approach

The development strategy follows a bottom-up approach, starting with fundamental components and progressively integrating and testing higher-level functions. Each component is validated through dedicated test cases (included in the *MainTest.m* script) that verify its performance individually before proceeding to system-level integration.

The flow of usage for the classes/functions is illustrated in the figure below.



In the next section of the report, each of these classes will be analyzed individually.

# Component Descriptions and Tests

## AudioData Class

### Functionality

Handles audio data loading, storage, and normalization. It ensures data is appropriately scaled for processing.

|  |  |
| --- | --- |
| **Properties** | **Methods** |
| * **‘Data’**: stores the audio samples loaded from a file. * **‘SampleRate’**: stores the sample rate of the audio data. | * **function obj = AudioData(filepath)**→ It is the constructor of the class. It uses the function ‘*audioread’* to load audio data from the specified ‘*filepath’*. The loaded audio samples and their corresponding sample rate are stored in the ‘*Data’* and ‘*SampleRate’* properties, respectively. * **function obj = normalize(obj)→** normalizes the audio data stored in the ‘*Data’* property so that the maximum absolute amplitude is exactly 1. |

### AudioData Test in MainTest

Checks file existence, loads audio, and normalizes it. The test verifies correct data loading and effective normalization by comparing pre and post-normalization amplitude levels.

**Command window output:**



## CustomFFT Function

### Functionality

The ‘customFFT’ function implements a recursive Fast Fourier Transform (FFT) algorithm tailored to input lengths that are powers of two (Cooley-Tukey algorithm). It showcases classic divide-and-conquer strategy used in FFT computations.

|  |  |
| --- | --- |
| **Input** | **Output** |
| * **x:** a vector representing the time-domain signal samples. The length of ‘x’ must be a power of two. | * **X**: a vector representing the frequency-domain spectrum of the input signal ‘x’. |

### Implementation Details

**Recursive Decomposition:** The function splits the input signal ‘x’ into two parts:

* X\_even: contains the samples from the even indices of ‘x’.
* X\_odd: contains the samples from the odd indices.

This step reduces the problem size by half, reducing the complexity. The recursion is applied to both ‘X\_even’ and ‘X\_odd’.



**Combination using the “Butterfly”:** After the recursive calls return the FFTs of the even and odd indexed parts, the results are combined using the “butterfly” operations typical of the FFT algorithms. This involves complex exponential multipliers, which are precomputed for efficiency.



**Recursion termination:**

* **Base Case:** The recursion terminates when the length of ‘x’ is 1, since the FFT of it is ‘x’ itself. This is the simplest form of the DFT, where the transform of a single sample is the sample.
* **Error Handling:** The function checks if the length of ‘x’ is a power of two. If not, it raises an error. This requirement is crucial in the **Cooley-Tukey algorithm**.

### CustomFFT Test in MainTest

Compares the output of **CustomFFT** against MATLAB's built-in **fft** function, ensuring the custom implementation's accuracy through magnitude and phase comparisons.A graph of blue lines

Description automatically generated with medium confidence

A graph of blue lines

Description automatically generated with medium confidence

**Command window output:**



## STFTProcessor Function

### Functionality

The **‘STFTProcessor’** function calculates the STFT for a single channel signal. It includes the windowing, overlapping and FFT processes.

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| * **x:** input signal. * **fs:** sampling frequency of the audio signals. * **window:** window function applied to each frame of the STFT. * **overlap:** a scalar indicating the proportion of overlap between consecutive frames. * **nfft**: the number of FFT points to compute the STFT. | * **S:** STFT matrix. * **f:** frequency axis. * **t:** time axis. |

### Implementation Details

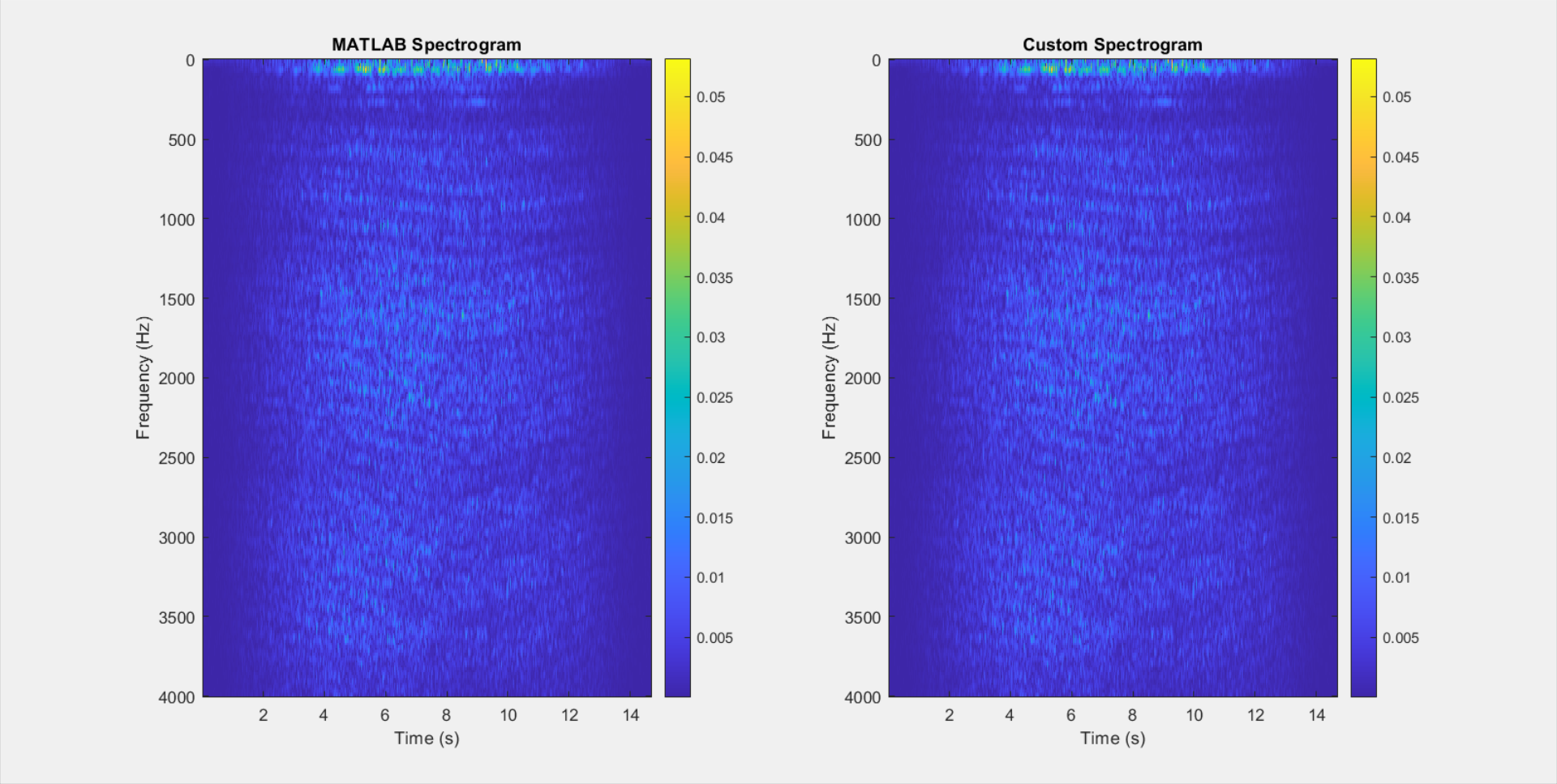
**Windowing and Overlap:** Computes the number of overlapping and applies the window to each frame of the signal.

**FFT Computation:** It uses the ‘customFFT’ function that handles padding and computes the FFT for each frame.

**Spectrum Handling:** It stores only the positive half of the frequency spectrum in the STFT matrix.

### STFTProcessor Test in MainTest

Verifies the custom STFT's correctness against MATLAB's **spectrogram** function by comparing the output matrices for discrepancies.



## AllChannelSTFT Function

### Functionality

Extends **STFTProcessor** to handle multichannel audio data, computing STFTs for each channel independently.

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| * **signal:** a matrix where each column represents an audio signal from one microphone. * **fs:** sampling frequency of the audio signals. * **window:** window function applied to each frame of the STFT. * **overlap:** a scalar indicating the proportion of overlap between consecutive frames. * **nfft**: the number of FFT points to compute the STFT. * **MicrophoneCount**: number of channels in the multichannel signal. | * **S\_multi**: a 3D array containing the STFT results for each channel. * **f:** the frequency axis for the STFT. * **t:** the time axis for the STFT. |

### Implementation Details

As showed in the code in figure X, the function performs the following operations:

**Channel Verification:** The function first checks if the number of columns in the ‘signal’ matrix matches ‘MicrophoneCount’ to confirm correct signal dimensions.

**STFT Calculation:** For each microphone channel, the function calls ‘STFTProcessor’ to compute the STFT.

**Output Initialization:** It initializes a 3D array ‘S\_multi’ after computing the first channel’s STFT to store all subsequent results.



### AllChannelSTFT Test in MainTest

Ensures that the function accurately processes each microphone channel without errors and integrates the results into a 3D array.

A group of yellow squares

Description automatically generated

## GetCovMatrix Function

### Functionality

The **‘**GetCovMatrix’function computes the spatial covariance matrix of signals captured by an array of microphones at different frequencies.

|  |  |
| --- | --- |
| **Input** | **Output** |
| * **S\_time**: a 2D matrix representing the STFT results for all microphones and frequencies at a specific time frame. | * **R**: a 3D array where each slice along the third dimension represents the covariance matrix for a particular frequency. |

### Implementation Details

As showed in the picture X, the function works in the following way:

**Frequency Iteration:** It processes each frequency band individually.

**Matrix Construction:** For each frequency, it calculates the outer product of the signal vector with itself to form the covariance matrix.



### GetCovMatrix Test in MainTest

Ensures that the matrices are Hermitian, affirming the correctness of the covariance computations.

**Command window output:**



## GetSteeringVector Function

### Functionality

The function ‘GetSteeringVector’ generates the steering vector for a microphone array, which is used in beamforming to focus the array’s sensitivity in a specific direction relative to the sound source.

|  |  |
| --- | --- |
| **Inputs** | **Output** |
| * **theta:** DOA angle in degrees. * **d:** spacing between microphones in the array. * **c:** speed of sound. * **numMics**: number of microphones in the array. * **freq**: frequency at which the steering vector is calculated. | * **a:** steering vector as a column vector. |

### Implementation Details

**Wavenumber Calculation:** It computes the wavenumber based on the frequency and the speed of sound.



**Vector Computation:** It uses the DOA angle and the microphone spacing to calculate the phase shift for each microphone, resulting in the steering vector.



## Beamform Function

### Functionality

Implements beamforming to enhance the signal from a specific direction using the steering vectors. It aggregates the energy from different directions to pinpoint the sound source's location. The function takes as input a 3D matrix (containing the STFT results for each frequency, time instant, and microphone) and returns a 3D matrix with 3 types of mean (arithmetic, harmonic, and geometric) of the beamforming results.

|  |  |
| --- | --- |
| **Inputs** | **Output** |
| * **S**: a 3D array containing the STFT results obtained in the function ‘AllChannelSTFT’  (called S\_multi). * **d**: the spacing between microphones in the array, measured in meters. * **c**: the speed of sound. * **Fs**: the sampling frequency. * **numMics**: number of microphones in the array. * **theta\_range**: array of angles (in degrees) for which the DOA estimates are computed → from -90° to 90°. | * **p\_theta\_time**: 3D matrix where each "page" (third dimension) represents a different type of mean:   + Page 1: Arithmetic mean   + Page 2: Harmonic mean   + Page 3: Geometric mean |

### Implementation Details

**Parameter Initialization:** It extracts the number of frequencies and time frames from the input data, and it also initializes the frequency vector.



**Time Frame Processing:** For each time frame, it extracts the corresponding STFT data for all frequencies and microphones.



**Covariance Matrix Calculation:** It computes the spatial covariance matrix for the signals received by the microphone array. The covariance matrix is computed calling the function ‘GetCovMatrix’.



**Angle Processing:** For each angle in ‘theta\_range’, the function computes the beamforming power using a steering vector, which is computed calling the function ‘GetSteeringVector’.



**Means Calculation :**

* **Arithmetic Mean**: Calculated as the simple average of all frequencies for each angle and time frame.
* **Harmonic Mean**: Calculated using the inverse of the average of the inverses.
* **Geometric Mean**: Calculated using the logarithmic sum to handle very small numbers.

% Compute the average power across frequencies

p\_theta\_time(:,:,1) = mean(p\_theta\_time\_freq, 3); % Arithmetic mean

% Compute the harmonic mean across frequencies

harmonic\_mean\_inv = mean(1 ./ p\_theta\_time\_freq, 3);

p\_theta\_time(:,:,2) = 1 ./ harmonic\_mean\_inv; % Harmonic mean

% Compute the geometric mean across frequencies

geometric\_mean\_log = sum(log(p\_theta\_time\_freq + 1e-10), 3) / numFreqs;

p\_theta\_time(:,:,3) = exp(geometric\_mean\_log); % Geometric mean

**Normalization:** At the end the function normalizes the power values across all angles and time frames to facilitate comparison.



### Test in MainTest

Covered implicitly through **DOAEstimator** tests, validating beamforming effectiveness in estimating directions.

## DOAEstimator Function

### Functionality

The ‘DOAEstimator’ function is designed to determine the DOA of sound based on the beamforming output computed in the ‘Beamform’ function. This function identifies the angle at which the received power is maximized, suggesting the most likely direction from which the sound originated. In addition, the function finds the DOA estimates for the 3 types of mean.

|  |  |
| --- | --- |
| **Inputs** | **Output** |
| * **p\_theta\_time**: a matrix (number of angles x number of time frames) containing the power values for each angle and time frame. This matrix is the output from the beamforming process, where each row corresponds to a different DOA angle, and each column corresponds to a different time frame. * **theta\_range**: an array of DOA angles (from -90° to 90°) → it corresponds to the rows in ‘p\_theta\_time’. These angles represent the possible directions from which the sound may arrive. | * **doa\_estimates**: A 2D matrix containing the estimated DOA for each time frame and each type of mean. Each row corresponds to a different mean. |

### Implementation Details

**Parameter Initialization**: It initializes an array ‘doa\_estimates’ to store the DOA estimates for each time frame. The length of the array must be equal to the number of time frames.



**Time Frame Analysis:**

* **Loop through Means**: The outer loop (using *meanIdx*) iterates through each type of mean (or page in the 3D matrix).
* **Loop through time frames**: the function iterates over each time frame within the current mean to analyze the power distribution across different angles.
* **Maximum power detection**: for each time frame, the function identifies the index of the maximum power in the ‘current\_p\_theta\_time’ matrix.
* **Angle association**: the angle corresponding to this index is retrieved from ‘theta\_range’ and recorded as the DOA estimate for that time frame.



### Test in MainTest

Integrates tests for **Beamform**, **GetSteeringVector**, and **GetCovMatrix** by using them to calculate the DOA estimates over time of a sweeping sound source that moves from 45 degrees to -45 degrees relative to the array's front, verifying the end-to-end functionality of the DOA estimation process.

A screenshot of a computer generated image

Description automatically generated

A graph with a line

Description automatically generated

## Visualization Functions (VisualizePseudospectrum and VisualizeDOAestimates)

These functions graphically represent the results of the DOA estimation process, providing a visual interpretation of how DOA varies over time and the pseudospectrum of detected angles. The functions that end with the work ‘Multi’ allow to see all the 3 version of the plot for different types of mean.

## Video Generation (getSingleFrame, FramesGenerator, VideoGenerator)

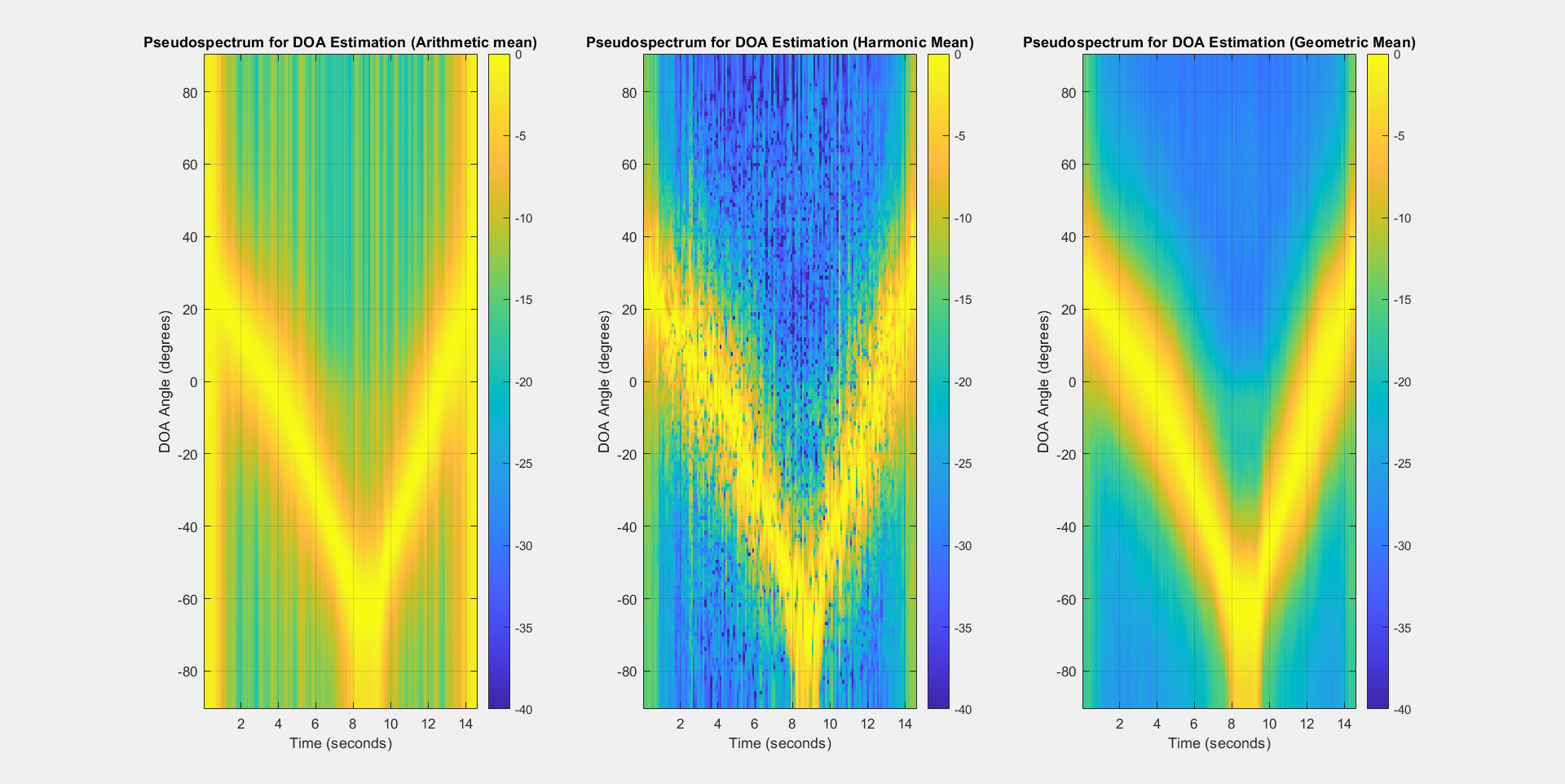
These functions generate visual frames and compile them into a video, illustrating the DOA estimation process over time. They create a dynamic visualization of the estimated angles as they evolve, enhancing understanding and presentation of the results.

## Main Script Functionality

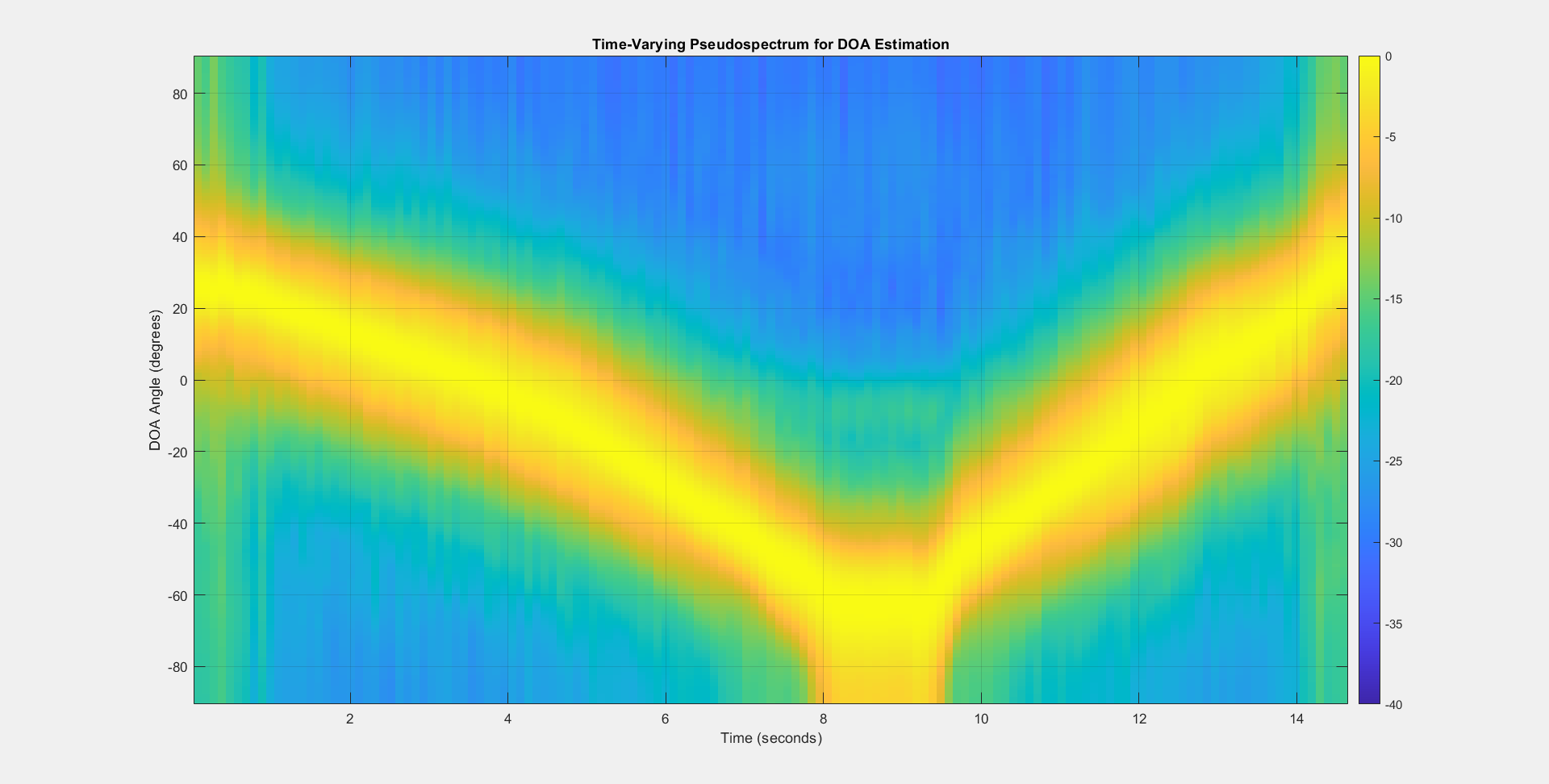
The main script orchestrates all the components to implement the DOA estimation process. It loads audio data, performs STFT, applies beamforming and DOA estimation, and visualizes the results through plots and video.

# Results Of The Provided Signal Analysis

## Analysis of the Pseudospectrum

The "Time-Varying Pseudospectrum for DOA Estimation" graph provides a comprehensive view of the power distribution across various directions (DOA angles) over time, allowing us to visualize how the direction of arrival of sound energy varies as the source moves. In the image below, we can see different pseudospectrums for different types of mean.

The best result we obtain is with the geometric mean since it is cleaner. Here is a bigger image of the plot :

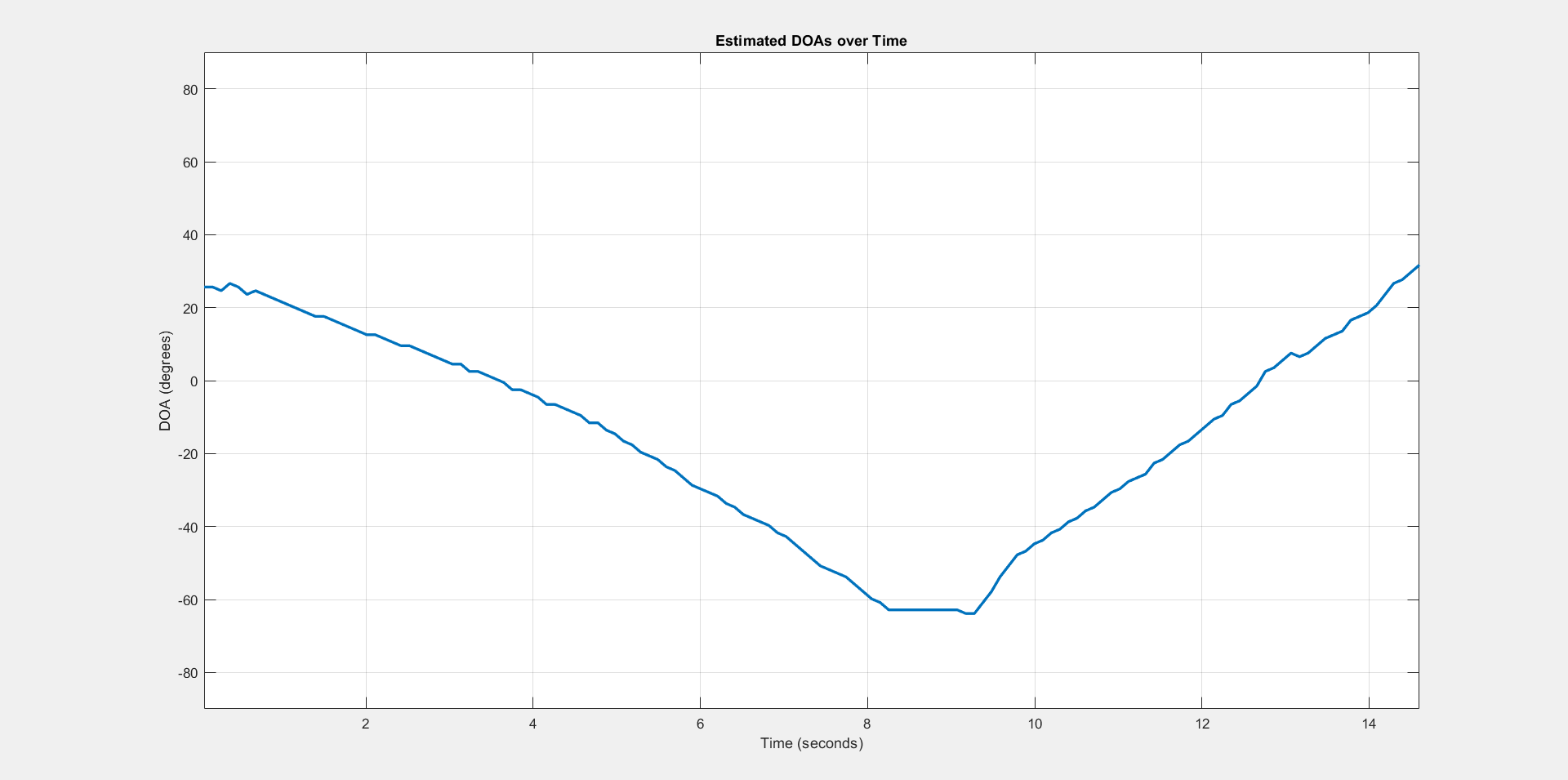


The color scale on the right represents energy levels, with warmer colors like yellow and orange indicating higher concentrations of energy and cooler colors such as blue representing lower energy levels.

The heatmap shows significant variations in energy concentration across the DOA angle and time. Notably, the areas of higher energy are found from +20 degrees, decreasing towards below -60 degrees, before gradually rising back towards the original position as time progresses from 0 to 14 seconds. This pattern suggests dynamic changes in the direction of arrival of the signal over the time interval displayed.

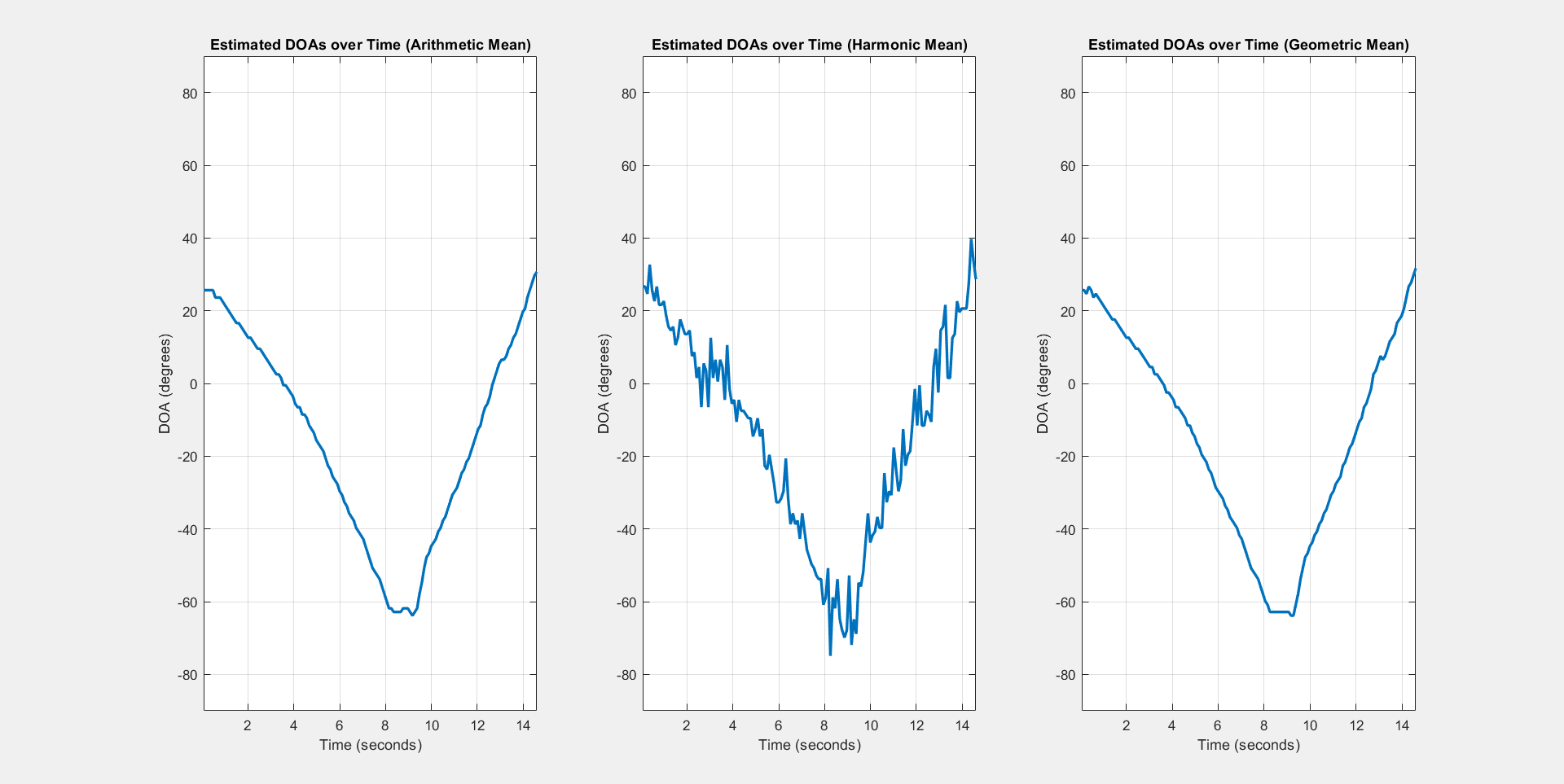
## Analysis of DOA Estimates

The graph below presents the estimated Direction of Arrival (DOA) of the sound source:



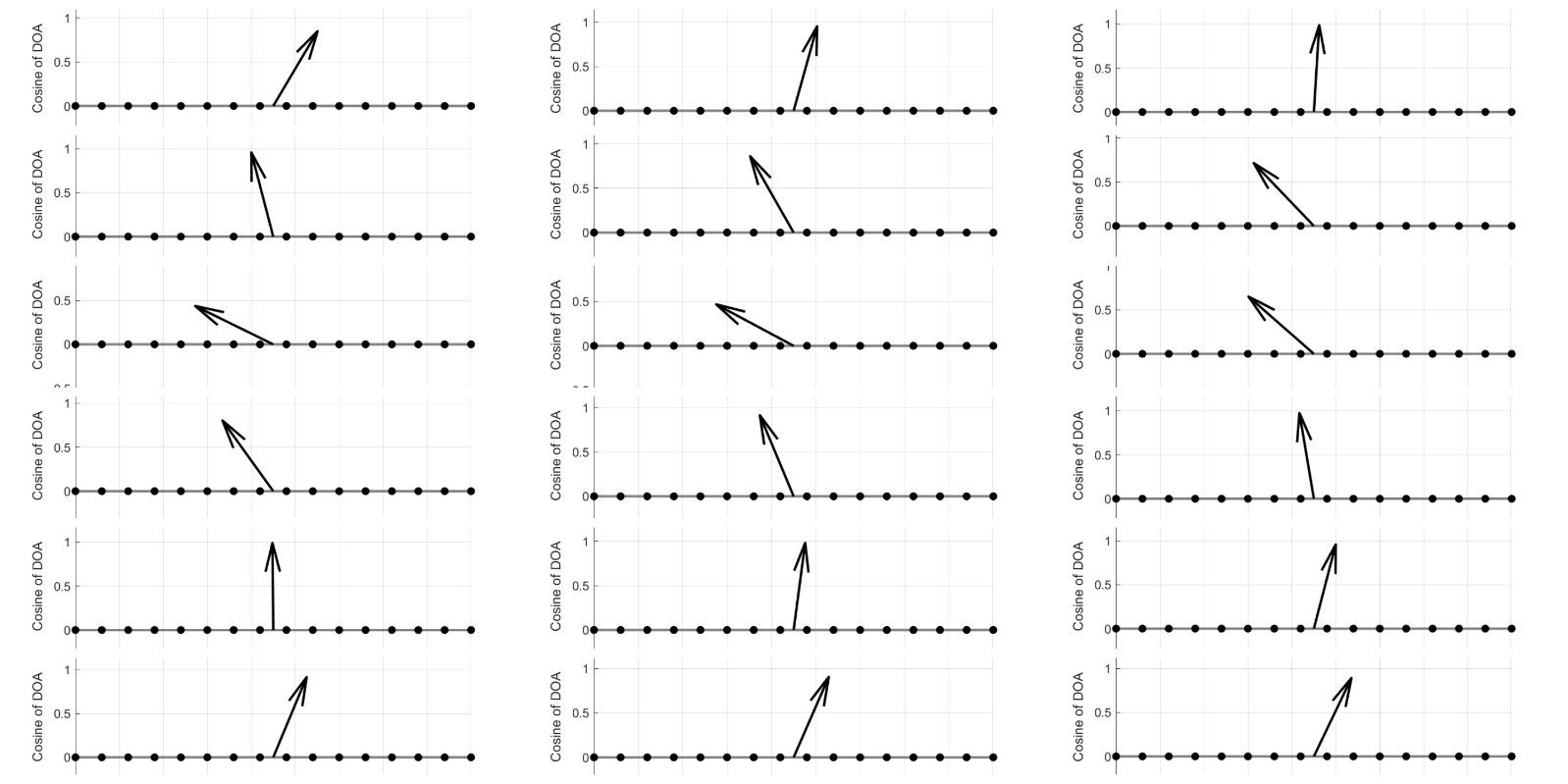
Since the sound source is moving, the DOA estimates plot displays that the DOA starts at a higher angle, drops to a significant low and then rises again. The graph begins at around 20° and experiences a steady decline to below -60°. The lowest point around the 100th sample suggests a moment where the sound source is at its farthest position relative to the initial orientation of the microphone array.

After the lowest point, there is a sharp recovery in the DOA estimates, where the angle swiftly increases, possibly indicating that the sound source is moving back towards its original starting direction.

We find the DOA estimates also with different type of mean, like for the pseudospectrum. Here are the result we obtained:

As we can see, using the harmonic mean, there a lot of jumps in the estimates plot, while with the arithmetic and geometric mean the trend is smoother.

We can also visualize the DOA estimates through the following image, which depicts a sequence of some frames from the video generated by the VideoGenerator function. This frame-by-frame visualization allows us to observe the temporal evolution of the direction of arrival estimates, highlighting how the system dynamically tracks the position of the sound source over time.



## Conclusion

In conclusion, the analysis results affirm the effectiveness of the designed acoustic source localization system.