HANOI UNIVERSITY OF SICENCE AND TECHNOLOGY

## SCHOOL ELECTRICAL AND ELECTRONIC ENGINEERING



**ASSIGNMENT 1 REPORT**

**Group 13**

**Subject** Multimedia Data Compression and Coding

**Instructor** Dr. rer. Nat Tien Pham Van

**Members** Pham Danh Tuan Dung 20233844  
Vu Duc Minh   
20233868  
Nguyen Minh Huy 20233852  
Luu Tien Dat  
2023

# Part I: Displaying the Spectrum of the Recorded Audio and Analyzing Energy Distribution

**Spectrum and Energy distribution :**

- The spectrum of an audio signal represents how its energy is distributed across different frequencies and is crucial for analyzing the characteristics of sound. It provides insight into which frequencies are present and how strong they are, serving as the foundation for many audio features, including MFCCs.

- Energy distribution refers to how the signal’s power is spread over time and frequency, helping to capture patterns such as speech intensity, silence, or emphasis. Together, spectrum and energy distribution are essential for understanding and interpreting audio signals in various applications like speech and music analysis.

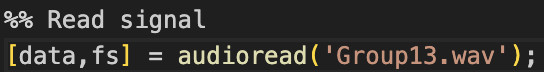
**Objective:**

The objective of this task is to visualize the frequency spectrum of a recorded audio clip and provide a brief analysis of its energy distribution across the frequency axis. This process helps us understand the dominant frequencies and spectral characteristics of the signal.

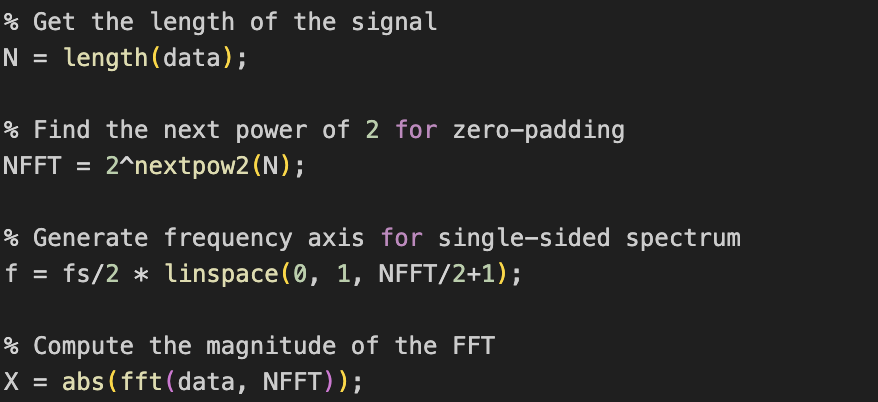
**Step-by-Step Process & Code Explanation:**

## Reading the Audio File

* + The first step is to read the audio file using the **audioread** function. This extracts the signal (**x**) and the sampling frequency (**fs**) from the audio file.

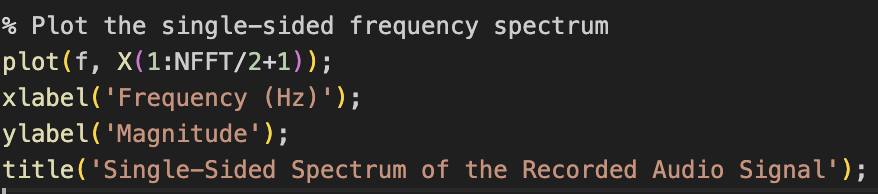


## Computing the Spectrum with FFT

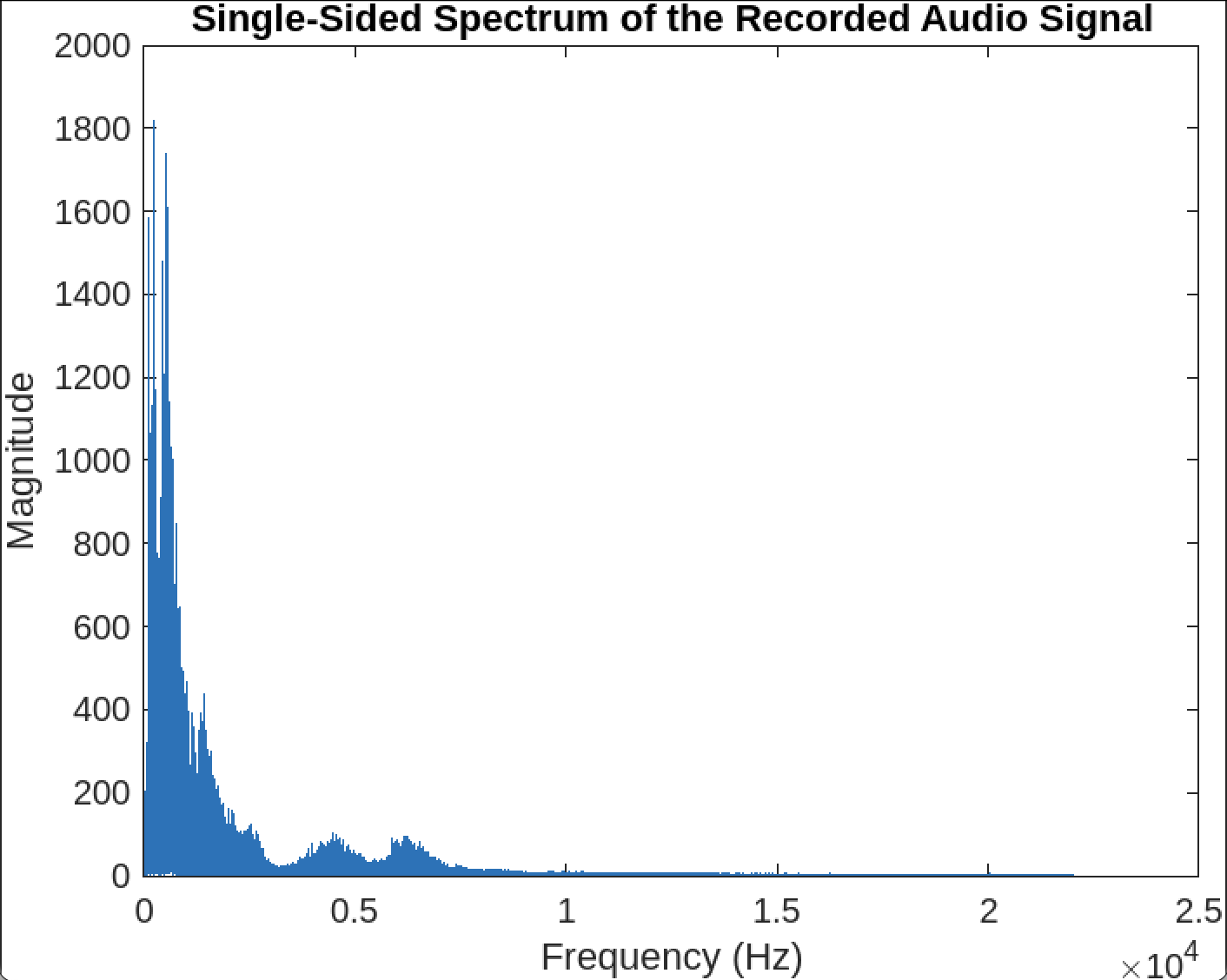
* + To analyze the frequency content of the signal, we apply the **Fast Fourier Transform (FFT)**:
* fft(data, NFFT) transforms the time-domain signal into the frequency domain.
* We compute only the **single-sided spectrum** because the input is a real-valued signal and its spectrum is symmetric.
* X represents the magnitude of frequency components.

### Visualizing the Frequency Spectrum

-We plot the single-sided spectrum to visualize energy distribution:



-This plot shows how much energy exists at each frequency component of the recorded signal.



## Comments on the Frequency Spectrum of the Speech Signal

* **Energy concentrated in low frequencies:**

The spectrum shows that most of the signal energy is concentrated in the range from **0 to around 4 kHz**. This is typical for human speech, where the most important phonetic information resides in the lower frequency bands.

* **Amplitude drops sharply with frequency:**

As the frequency increases, the amplitude decreases rapidly. This indicates that the higher frequency components (above 5–6 kHz) carry very little energy, which is expected for speech signals.

* **Reasonable frequency range:**

The frequency axis extends up to about **24 kHz**, corresponding to a sampling rate of **48 kHz**. However, since speech primarily occupies the range below 8 kHz, the spectrum beyond 10 kHz is nearly flat or zero.

### Conclusion:

The speech signal shows a typical spectral pattern where energy is concentrated in the **0–4 kHz** range, while higher frequencies contain minimal energy. This is useful for applications such as audio compression (to reduce file size without losing important information) or speech transmission (where only the necessary frequency band is needed).

# Part II: Audio Compression Using MFCC with Warping Codec (PCA)

**MFCC with Warping: Overview :**

MFCCs (Mel-Frequency Cepstral Coefficients) are a compact representation of the power spectrum of audio signals and are widely used in tasks like speech recognition, speaker identification, and audio classification. To account for variations in speaker characteristics—such as vocal tract length—frequency warping techniques are applied during MFCC extraction. Common approaches include Vocal Tract Length Normalization (VTLN), which applies a warping function to normalize frequency scales; Bark or Mel warping, which use fixed perceptual scales; and custom warping functions that can be manually defined or learned from data.

**Objective :**

The goal of this task is to compress an audio signal using Mel Frequency Cepstral Coefficients (MFCC), combined with a Warping codec based on Principal Component Analysis (PCA). This method reduces the dimensionality of the audio’s feature representation and then attempts to reconstruct the signal from the compressed features.

**Step-by-Step Process & Code Explanation:**

1. **MFCC with warping**

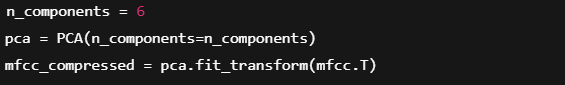
* **Step 1: Load original audio**

****

* + Loads the audio file as a waveform (y) and gets the sampling rate (sr).
  + sr=None ensures the original sample rate is preserved.
* **Step 2: Extract MFCC features**



* + Extracts 13 MFCC (Mel-Frequency Cepstral Coefficients) from the audio.
  + These features represent the timbral texture of the sound.
* **Step 3: Compress MFCC using PCA (simulate a Warping codec)**



* + Transposes the MFCC matrix to shape (frames, 13) before applying PCA.
  + Reduces the 13-dimensional MFCC features to **6 principal components**.
  + This simulates compression.

### Step 4: Decompress (Restore MFCC)



* + Reconstructs the MFCC matrix from the compressed version.
  + Transposed back to the original shape (13, frames).

### Step 5: Reconstruct audio from MFCC



* + Converts the MFCC features back into an audio waveform.
* **Step 6: Save the reconstructed audio**

****

* + Writes the reconstructed audio to a .wav file.

**Conclusion:**

- The MFCC + PCA method provides a lightweight, perceptually relevant representation of audio.

- While it does not preserve perfect fidelity like MP3, it is highly suitable for tasks where **dimensionality reduction and feature preservation** are more important than exact signal reconstruction — such as speech recognition or audio classification.

- This approach demonstrates how audio compression can benefit from signal processing and machine learning techniques.

# Part III: Comparison of Compressed Audio Quality Using PSNR

**PSNR :**

PSNR (Peak Signal-to-Noise Ratio) is a metric used to measure the quality of a reconstructed or compressed signal compared to its original version. It is commonly used in image and audio processing to evaluate how much distortion or noise has been introduced. Expressed in decibels (dB), a higher PSNR value indicates better quality and closer similarity to the original signal, while a lower PSNR suggests more noticeable degradation.

**Objective:**

The objective of this task is to evaluate and compare the quality of two audio compression methods:

1. MFCC-based compression using Principal Component Analysis (from Task 3)
2. MP3 compression using standard lossy audio codec

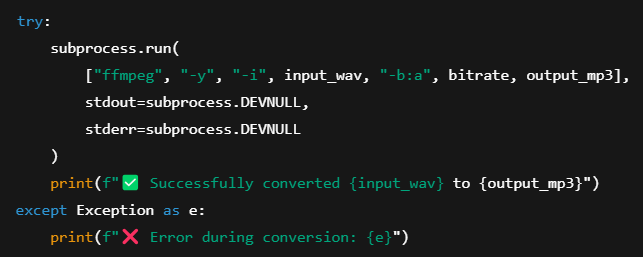
The quality of the reconstructed audio from each method is assessed using the Peak Signal-to-Noise Ratio (PSNR).

### Convert a WAV file to an MP3 file using Python and ffmpeg

* **Step 1: Define the conversion function**



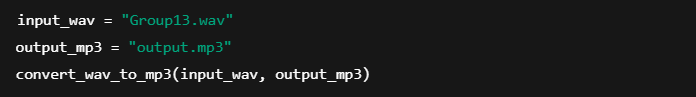
* + Defines a function to **convert a WAV file into an MP3 file** using the specified bitrate.
  + Parameters:
    - input\_wav: path to the input WAV file.
    - output\_mp3: path to save the output MP3 file.
    - bitrate: optional, default is "128k"
* **Step 2: Handle the conversion with try block**



* + **Try block** runs the ffmpeg command:



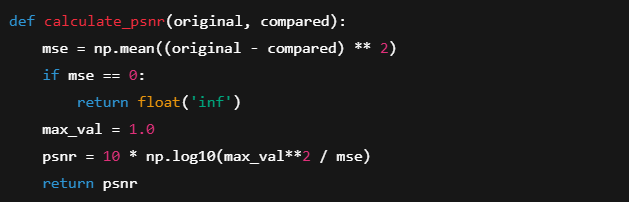
* + -y: automatically overwrite the output file if it exists.
  + -i: specifies the input file.
  + -b:a: sets the audio bitrate.
  + stdout and stderr are suppressed (no console output).
* **Step 3: Example usage**



* + Converts Group13.wav to output.mp3 with 128 kbps bitrate.

1. **Compare the quality of the compressed sound to MP3 compression, showing PSNR statistics.**

* **Step 1: Define PSNR calculation function**



* + **PSNR** measures how close two signals are (higher PSNR = better quality).
  + If MSE (Mean Squared Error) is 0 → audio is identical → PSNR is ∞.
* **Step 2: Load original WAV audio**



* + Loads the original uncompressed WAV file.

### Step 3: Load MFCC-reconstructed audio

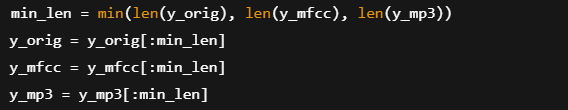


* + Loads audio that was compressed and reconstructed using MFCC + PCA.

### Step 4: Load MP3-compressed audio



* + Loads the MP3 version of the original file.
* **Step 5: Trim audio to the same length**

****

* + Makes sure all three signals are the same length before comparison.

### Step 6: Calculate PSNR values



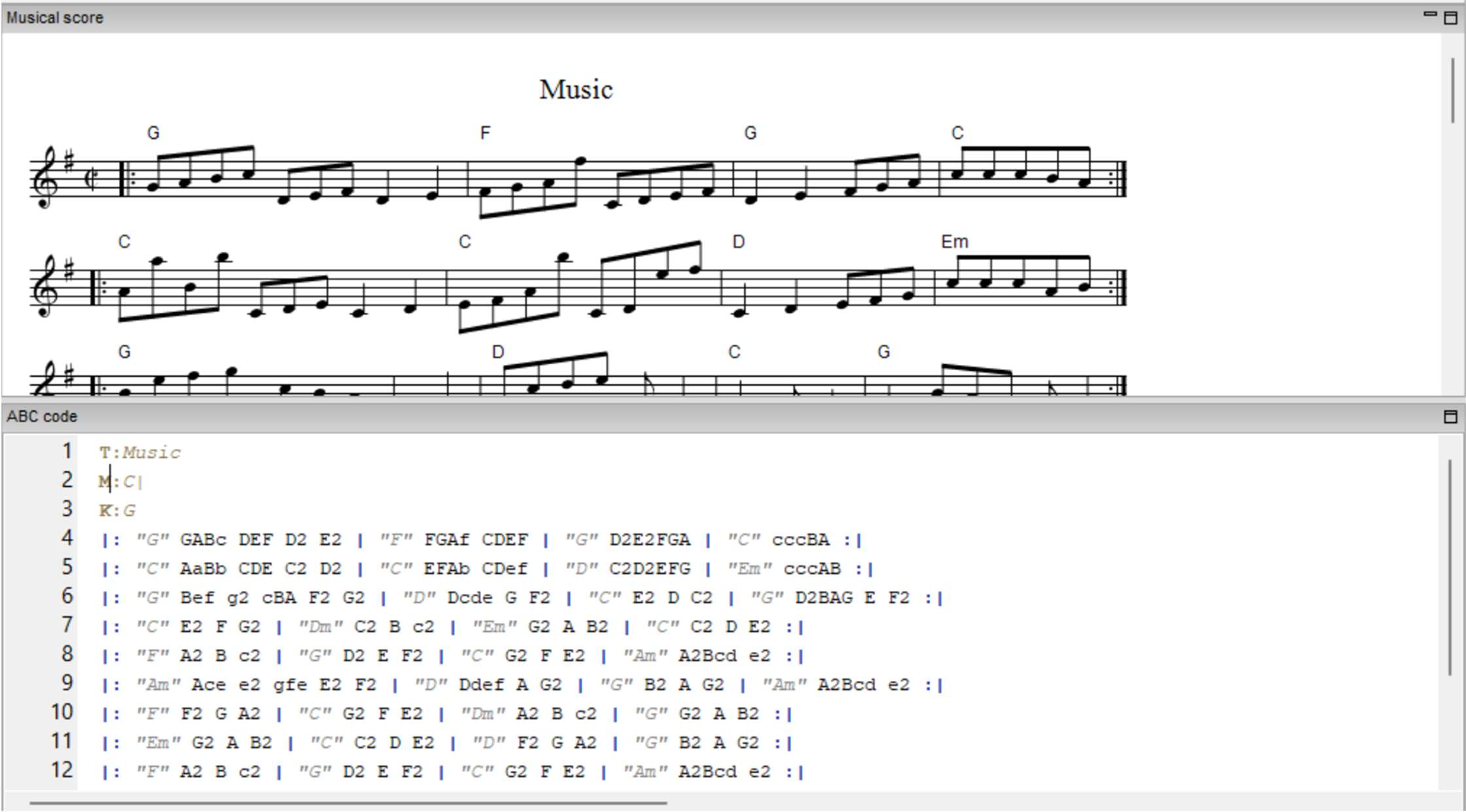
* + Calculates how much the MFCC and MP3 versions deviate from the original.

# 

**Part IV: MIDI compilation and music mixing**

## MIDI compilation

* + First, we create a .abc file using the ABC editor EasyABC.

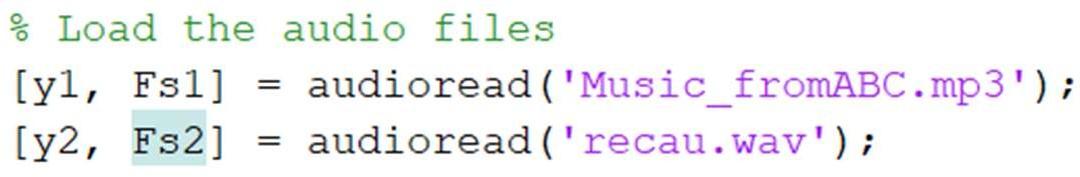


* + The program has a nice interface, allows us to create music with ABC code. Besides music notes, we can modify the rhythm, duration, and other musical elements.
  + We exported the music in MIDI format.

## Mixing music

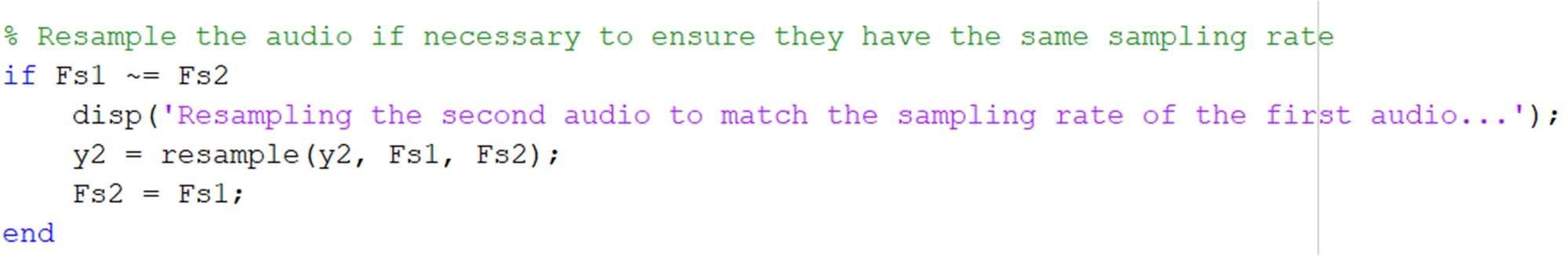
* + The steps to mix the music with recorded audio:

1. Read two files



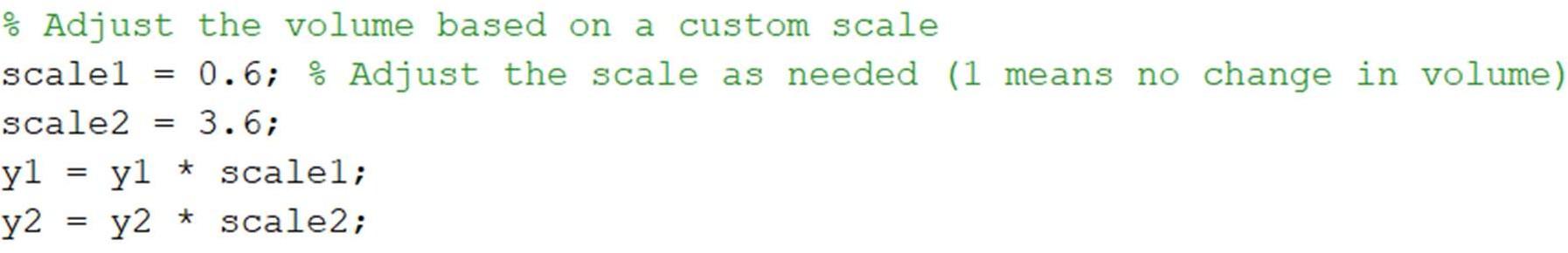
* + The code reads the audio files along with their respective sampling frequencies.

1. Resample Audio



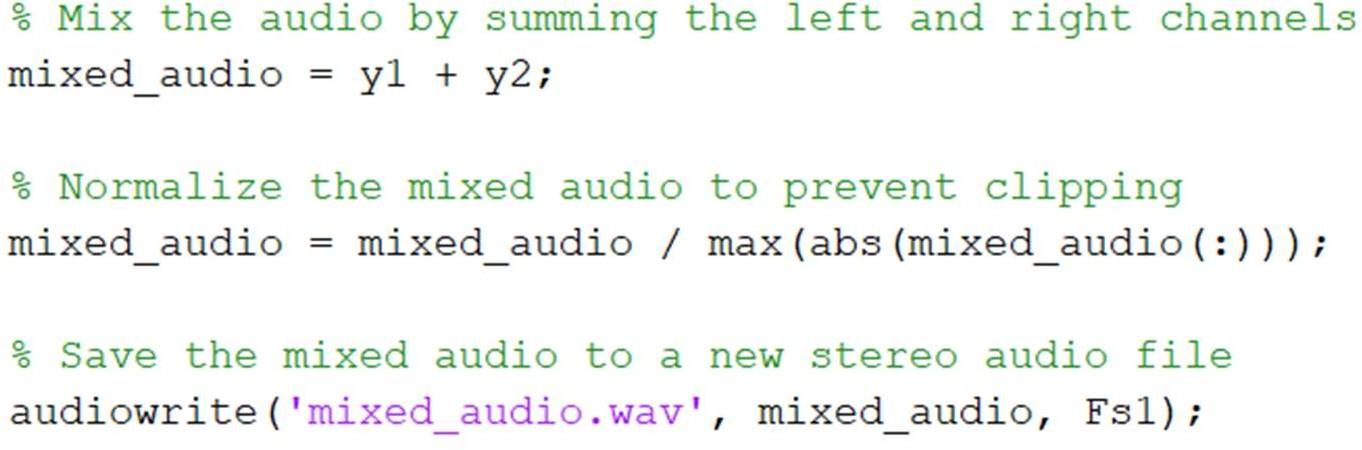
* + If the sampling frequencies of the two audio files are different, the code resamples the lower sampling frequency audio to match the higher one using the **resample** function.

1. Adjust Volume



* + Adjust the volume of each audio file by multiplying their data with their respective scaling factors.

1. Mixing



* + Write the mixed audio data using the **audiowrite** function.
  + We mixed the recording audio with the music created from EasyABC for better result.