# TSKS37 Lab 2
# Audio Processing

Fall 2024
Version of this document: January 17, 2025

**Examination**

This exercise is a set of tasks described further down in the document. During the oral examination, we expect you to present the required figure(s) and/or sound(s). The lab assistant may also ask for an explanation of the written code or questions highlighted in bold characters.

There is no requirement on using jupyter for this lab, and thus you can use either a jupyter notebook or regular `.py` files.

**Code Submission**

You don't have to do anything now. After completing the third lab, you will submit the code from both this and the third lab at the same time (following the instructions provided in the third lab).

## 1 Audio Processing in Python

This exercise aims to introduce audio processing in Python. You will learn how to load, play and work with audio files, as well as how to analyze and visualize some important characteristics of audio signals.

### 1.1 Main tasks

1. Mix two audio sources.
2. Add noise to the mixed signal.
3. De-mix the combined signal and observe how the noise level degrades the quality.
4. For the different signals above, visualize the sound in both the time domain (waveform) and the frequency domain (Spectrogram). Pay attention to the style guidelines introduced in the first lab (see the `plot.pdf` file). **All figures must**: clearly visualize the data, have reasonably sized labels, appropriate units, legends if needed, and be legible in gray-scale print.

### 1.2 Libraries

Throughout the lab you will need to use the following Python libraries:

- numpy,
- matplotlib,

- scipy - load audio files to a format we can work with (numpy arrays),
- pydub - listen to audio files.

In these instructions, we will recommend the usage of some functions from these libraries. You **will have to read the documentation** for the different functions to learn how to use them properly.

All the required libraries will be installed on the lab computers. If you want to run the code on your own computer you need to pay attention to a couple of things:

- scipy can be installed using pip through the call *pip install scikit-learn* (note the name),
- To use pydub you will need to have *ffmpeg* installed as well. Note that ffmpeg can't be downloaded through pip. A compiled version can be downloaded from https://ffmpeg.org. [1]

### 1.3 Hardware

Headphones will be provided to you during the lab sessions (ask the lab assistant). You must use them, or personal headphones, when working in the lab room.

## 2 Mathematical Preliminaries

Given two length $L$ audio vectors $x_1$ and $x_2$, we define the $L \times 2$ combined audio matrix as $X = [x_1, x_2]$. This is a matrix with $x_1$ in its first column and $x_2$ in its second column. The mixed audio matrix is $Y = XW$ for some $2 \times 2$ weight matrix $W$. Each of the two columns of $Y$ represent one mixed audio vector of the same length as $x_1$ and $x_2$. We will create the weight matrix as follows:

$$W = \begin{bmatrix} \alpha & 0 \\ 0 & 1-\alpha \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \alpha\cos(\theta) & -\alpha\sin(\theta) \\ (1-\alpha)\sin(\theta) & (1-\alpha)\cos(\theta) \end{bmatrix}, \tag{1}$$

where $\alpha \in [0,1]$ and $\theta \in [0, \pi/2]$ are our two mixing parameters. $\alpha$ divides the available power between the two audio sources whereas $\theta$ describes to what degree they are mixed together. You will experiment with these parameters later in the lab.

After creating the mixed matrix we will add some random noise to it. This is done by adding an $L \times 2$ noise matrix $N$ to $Y$, giving us $Z = Y + N$. Details on how you will construct the noise matrix are given in Section 2.2.

Using this noisy version of the mixed audio matrix ($Z$) we can then attempt to recreate the original audio signals through the process called demixing. We get an estimate of the combined audio matrix by multiplying with the inverse of the weight matrix:

$$\hat{X} = [\hat{x}_1, \hat{x}_2] = ZW^{-1} = X + NW^{-1}, \tag{2}$$

where $NW^{-1}$ is the (effective) error term. Note that in the case with no noise, i.e., $N = 0$, demixing exactly recovers the original audio signals (as long as $W$ is invertible). Since $W$ is a $2 \times 2$ matrix the inverse can for example be computed according to:
$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$.
Note that choosing $\alpha = \{0, 1\}$ leads to $W$ being singular (i.e., not invertible). Therefore, when

[1] In the download section you will find "Get packages & executable files". After downloading this you will have a folder in which there is a folder called "bin". The only thing you need to do is to add this bin folder to your path.

experimenting with parameter values in the lab, you should use values close to zero and one rather than exactly those values (study the asymptotic behavior of demixing).

## 2.1 Error Computation

We will use the following metric to measure the error of the reconstruction:

$$error = \frac{||\hat{X} - X||_F}{||X||_F} = \frac{||NW^{-1}||_F}{||X||_F}, \tag{3}$$

where the so called Frobenius Norm $||A||_F = \sqrt{\sum_{i,j} |A_{i,j}|^2}$. That is, we compare each individual element in $\hat{X}$ with the respective element in $X$ and sum up the square of these differences for all elements.

## 2.2 Noise Generation

As mentioned earlier, we need to generate some random noise. For that, the concepts of random variables and distributions are essential. These are things that you will come across in many future courses but for now, it is sufficient that you know that we will use the so-called *Gaussian* (or normal) distribution to create the noise.[2] The distribution is specified through two parameters, the *mean* $\mu$ and the *variance* $\sigma^2$. We will use $\mu = 0$ throughout the entire assignment. Then, the variance corresponds to the *power* of the noise. The larger the variance, the more noise we have. The variance will be set to achieve a specific so-called signal-to-noise-ratio (SNR). That is, a ratio between the power of the signal (defined as the square of the Frobenius norm of the signal) and the power of the noise (the variance of the Gaussian distribution) [3]. The SNR can thus be computed as

$$SNR = \frac{||X||_F^2}{\sigma^2}. \tag{4}$$

We often give the SNR in dB rather than in a linear scale as above. Conversion between dB and linear scale can be done as $\text{SNR}_{dB} = 10\log_{10}(\text{SNR}_{\text{linear}})$.

[2]You might be familiar with the concept of a "bell curve" which is exactly the normal distribution.

[3]In reality, the noise power is not something a user of a system can control. We would instead have to increase the signal power enough to achieve an acceptable SNR value.

## 3 Exercises

### 3.1 Listen to the audio files

In this assignment, we will work with two different 3-second audio segments which can be found in the files "CantinaBand3.wav" and "StarWars3.wav". The first task is to listen to them and for this, we will use the pydub library. First, we need to load the audio segments into Python which can be done using the pydub.AudioSegment.from_wav() function. This returns an object that can be passed to the pydub.playback.play() function to listen to the audio. You will later be asked to listen to different processed versions of these sounds and compare them to the original audio so it is recommended that you create a function that does this.

### 3.2 Load and mix the audio files

The object created above is not convenient to use for processing the signals. We will instead load the data using the scipy library which will give us representations of the audio signals as numpy arrays.

(a) Load the two audio files using scipy.io.wavfile.read(). It returns the *sample rate* and a numpy array with the data. **What unit is the sample rate given in**, and **what are the dimensions of the data arrays?**
Now, create a figure consisting of four subplots in a $2 \times 2$ grid. Plot the two *waveforms* (arrays) in two of these subplots. In the other's you will plot their *spectrograms* which is a representation of the frequency content of a signal. You do this by plotting the absolute value of the Fourier transform of the signals against the sample frequencies [4]. scipy.fft.fft() can be used to compute the Fourier transform and scipy.fft.fftfreq() for the sample frequencies. For scipy.fft.fftfreq() you will have to manually set two of the parameters "n" and "d". Note that the "window length" mentioned in the documentation is the length of the audio vectors in our case.

(b) We will now mix the two signals together. Following the mathematical description in section 2, mix the two signals using weight matrix $W$. The result is two separate audio tracks that both are combinations of the two original ones (one in each column of the mixed matrix). In order to listen to them you need to first save them as .wav files which can be done using scipy.io.wavfile.write(). Here you need to pay attention to two things: 1) you should save each of the tracks in a separate file, and 2) before saving them as .wav files you need to convert them to int16 arrays using for example the function ndarray.astype(numpy.int16). Listen to them following the same steps as in exercise 1. **Try some different values of $\theta$ and $\alpha$ to see how the sound changes. Can you find values so both songs are clearly audible and so that only one is (in either of the tracks)? What happens when $\theta = 0$ and $\theta = \pi/2$? What happens when $\alpha$ approaches 0 and 1? Why?**

(c) Use a mixed matrix created with $\alpha$ and $\theta$ values such that both songs are clearly present in both tracks. Add Gaussian noise to the mixed matrix using numpy.random.normal(). Make sure to create a noise matrix of the same dimension as the mixed matrix. Set the mean to zero and the variance to achieve SNR = 5. The required noise power to achieve a specific SNR can be computed using Eq. (4). The function numpy.linalg.norm() can be used to compute the Frobenius norm (power) of $X$. When using numpy.random.normal(), you need to pay attention to two things: 1) the function expects the standard deviation (squareroot of the variance) and 2) you will have the total power of the noise matrix but the function needs it per element. Thus the *scale* argument should be set to $\sqrt{\sigma^2/(\#\text{ elements in } X)}$. **Verify that you have the correct**

[4]The Fourier transform is another extremely important concept that you will come across in the future. Although you don't need to understand it for this course, we can recommend the following introductory video to the topic: https://youtu.be/spUNpyF58BY

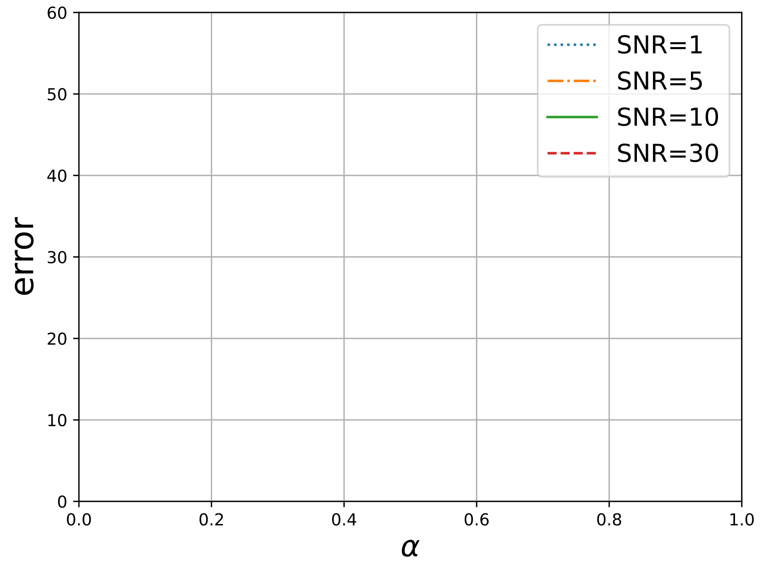

Figure 1: Example of how the figure should look (aside from the actual data).

**variance by computing the actual SNR** as $SNR = \frac{||X||_F^2}{||N||_F^2}$ and compare it to the target. [5]

Listen to the noisy mixed audio (one of the tracks is sufficient). **How does the SNR affect the sound? Try some different SNR values including some below one!**

(d) Lastly, choose one of the two tracks (columns) to plot. For that column, create a figure consisting of 3 subplots: 1) its waveform, 2) the waveform of the noise (only the corresponding column) and 3) the waveform of the noisy mixed audio.

### 3.3 Demix the audio signal.

We will now recreate the original soundtracks from the noisy mixed matrix.

(a) Demix the audio signals by multiplying the noisy mixed matrix with the inverse of $W$. Then, listen to the demixed audio signals and compare them to the originals. **How does the quality change with the SNR, $\alpha$ and $\theta$? Are their any parameter values for which reconstruction seem to fail?** Recreate the plot from exercise 2(a) with these (noisy) reconstructions and compare them. This plot will depend on $\alpha$ and $\theta$, **present at least two versions of the plot, one with $\alpha$ and $\theta$ values resulting in good reconstruction and one with bad reconstruction**.

(b) Finally, we will visualize the reconstruction error as defined in Section 2 and investigate how $\alpha$ and the SNR affect this error. To do this, we will first fix $\theta = \pi/3$. Then, for some different SNR values, generate curves of how the error changes as a function of $\alpha$ and plot them in the same figure. See Figure 1 for an example (where the actual curves have been left out). **Is there an optimal $\alpha$? If not, why not? And if yes, does the value make sense? Does your result align with your findings in 3(a)?**
Hint: you might have to adjust the y-limit in the plot to other values than in the example.

[5]Since the noise is random you will not get exactly 5 but it should be fairly close.