AUDIO SOURCE SEPARATION USING NON-NEGATIVE MATRIX FACTORIZATION

Priyansh Jain (210070063)
Ashish Prasad (21d180009)
Alok Kumar (210070006)
Anushka Nalawade (21B080007)

Introduction

The main focus of this project is the task of sound source separation, which consists in isolating individual sound sources from a mixture of sounds. This can be used in several applications, such as music separation, speech separation and speech enhancement.

In music separation, for example, we can separate the different instruments in a song, allowing the individual tracks to be edited or remixed. In speech separation, we separate different speakers in a recording, making it possible to transcribe the speech of a single speaker. Lastly, speech enhancement can separate the vocals and the background noise or song.
We will be extracting the audio sources by using NMF applied to the spectrogram of a short piano sequence. The project was done in Python.

Solving Approach

We will be using Non-Negative Matrix Multiplication (NMF). NMF algorithm seeks to reconstruct/ factorize a given data matrix V into two non-negative unknown matrices W and H, such that the product of the two matrices approximates the original matrix.

$$V \approx WH = \hat{V}$$

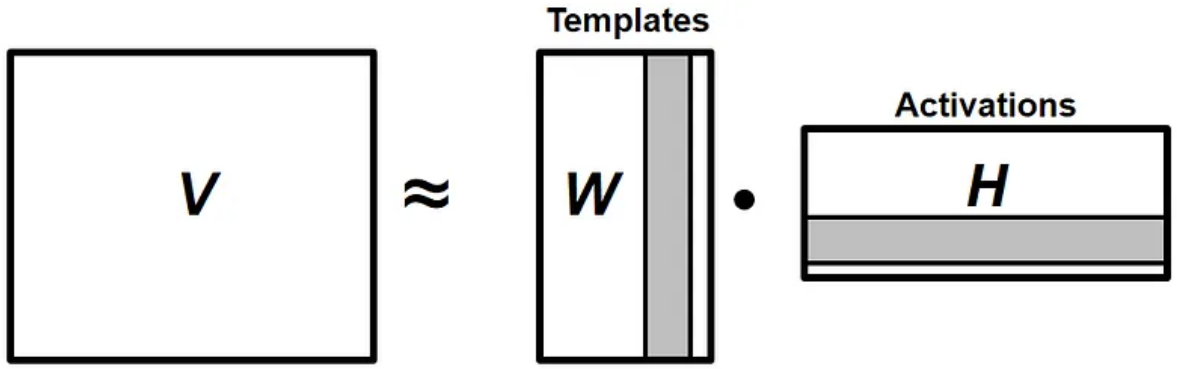
NMF is unique from other types of rank reduction algorithms as W and H must be strictly non-negative.

$$W \geq 0$$
$$H \geq 0$$

In sound source separation:

- The original data matrix V is the time-frequency representation of the audio signal, also known as a spectrogram.
- W represents a dictionary of elementary recurring features of the data.
- H contains the activations of the different features.

W is referred to as the dictionary matrix and H as the activation matrix.



Given V of dimension K x N. W and H, are of dimensions K x *S and S* x N, respectively, where S is the common dimension, that can be thought of as the number of sources we wish to extract and can be referred to as the rank of the factorization.

To obtain a good reconstruction of the signal's spectrogram, the estimation of W and H are obtained by optimizing a cost function using the iterative updates of a gradient descent.

This means we will try to find the W and H that minimizes the distance between the given data matrix V and the estimated one.

$$\arg\min_{W,H\geq 0} = D(V|WH) = D(V|\hat{V}) = \sum_{i}^{K}\sum_{j}^{N} d(V_{i,j}|[WH]_{i,j})$$

Cost Functions

The cost function used is called β divergences and defined as:

$$d_\beta(x|y) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{\beta(\beta-1)}\left(x^\beta + (\beta-1)y^\beta - \beta x y^{\beta-1}\right), & \beta \in \mathbb{R}\backslash\{0,1\} \\ x\log\frac{x}{y} - x + y = d_{KL}(x|y), & \beta = 1 \\ \frac{x}{y} - \log\frac{x}{y} - 1 = d_{IS}(x|y), & \beta = 0 \end{cases}$$

The quadratic cost can also be found for β = 2

$$d_Q(x|y) = \tfrac{1}{2}(x-y)^2$$

Gradient Descent Algorithm

We denote θ a parameter corresponding to either W or H. The update rule of the gradient descent algorithm looks like this:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta \beta Div$$

$$where : \theta^{(i+1)} \geq 0$$

Each gradient of the introduced cost functions can be written as a difference between two non-negative functions.

$$\nabla f(\theta) = [\nabla f(\theta)]^+ - [\nabla f(\theta)]^-$$

We will be using adaptive learning rates to avoid subtraction that can produce negative elements. Thus, the learning rate is set to :

$$\eta = \frac{\theta^{(i)}}{\nabla_H^+ \beta_{Div}}$$

When we replace the last two equations in the general gradient descent equation, we arrive at the given update rule :

$$\theta^{(i+1)} \leftarrow \theta^{(i)} \bigotimes \frac{[\nabla f(\theta)]^-}{[\nabla f(\theta)]^+}$$

This approach is taken such that the updates are multiplicative, where the correction value is equal to the ratio of the negative and positive parts of the derivative of the criterion.

The non-negativity of the multiplying update values ensures the non-negativity of the parameters W and H along the optimization iterations.

Computing the Gradients

Let's take the generalized beta divergence definition seen above:

$$\beta Div(V|WH) = \frac{1}{\beta(\beta-1)}(V^\beta + (\beta-1)[WH]^\beta - \beta V[WH]^{\beta-1}))$$

$$\beta Div(V_{i,j}|[WH]_{i,j}) = \frac{1}{\beta(\beta-1)} \sum_{ij} (V_{i,j}^\beta + (\beta-1)[WH]_{i,j}^\beta - \beta V_{i,j}[WH]_{i,j}^{\beta-1})$$

The gradient of this cost function can be computed in terms of each parameter W or H at a time, denoted as θ, as such :

$$\frac{\partial \beta Div(V|WH)}{\partial \theta_{p,q}} = \frac{1}{\beta(\beta-1)} \sum_{ij}((\beta-1)[WH]^{\beta-1}i,j\frac{\partial [WH]i,j}{\partial \theta_{p,q}} - \beta(\beta-1)V_{i,j}[WH]^{\beta-2}i,j\frac{\partial [WH]ij}{\partial \theta_{p,q}})$$
$$= \sum_{ij}([WH]_{i,j}^{\beta-1}\frac{\partial [WH]_{i,j}}{\partial \theta_{p,q}} - \beta V_{i,j}[WH]_{i,j}^{\beta-2}\frac{\partial [WH]_{i,j}}{\partial \theta_{p,q}})$$

We know by definition that each data point in the estimated product matrix of index [i,j] is the sum of the term-by-term multiplication of the ith row of W and the jth column of H.

$$[WH]_{i,j} = \sum_k W_{i,k}H_{k,j}$$

The partial derivative of [WH] over H for a given W :

As W is considered constant, we have :

$$\frac{\partial [WH]i,j}{\partial Hp,q} = \sum_k W_{i,k}\frac{\partial H_{k,j}}{\partial H_{p,q}}$$

This partial derivative is only non-zero for k = p and j = q, which can be translated to 2 Dirac delta function in those points. The sum over k can be removed since all terms will be equal to zero except when k = p.

$$\frac{\partial [WH]i,j}{\partial Hp,q} = \sum_k W_{i,k}\delta_{j,q}\delta_{k,p} = W_{i,p}\delta_{j,q}$$

The gradient of the cost function with respect to H for a given W :

if we substitute this result in the previous gradient equation :

$$\frac{\partial \beta Div(V|WH)}{\partial H_{p,q}} = \sum_{ij}([WH]_{i,j}^{\beta-1}\frac{\partial [WH]_{i,j}}{\partial H_{p,q}}) - \sum_{ij}(V_{i,j}[WH]_{i,j}^{\beta-2}\frac{\partial [WH]_{ij}}{\partial H_{p,q}})$$
$$= \sum_{ij}([WH]_{i,j}^{\beta-1}W_{i,p}\delta_{j,q}) - \sum_{ij}(V_{i,j}[WH]_{i,j}^{\beta-2}W_{i,p}\delta_{j,q})$$
$$= \sum_{i}([WH]_{i,q}^{\beta-1}W_{i,p}) - \sum_{i}(V_{i,q}[WH]_{i,q}^{\beta-2}W_{i,p})$$

We can then go back to a more simplified matrix form :

$$\frac{\partial \beta Div(V|WH)}{\partial H_{p,q}} = \sum_{i}(W_{p,i}^T[WH]_{i,q}^{\beta-1}) - \sum_{i}(W_{p,i}^T[(WH)^{\beta-2}V]_{i,q})$$
$$= W^T(WH)^{\beta-1} - W^T[(WH)^{\beta-2}V]$$

Finally, we get a gradient that can be written as a difference between two non-negative functions.

The gradient of the cost function with respect to W for a given H:

The same process can be used to compute the solution over W to get :

$$\frac{\beta Div(V|WH)}{\partial W_{p,q}} = (WH)^{\beta-1}H^T - [(WH)^{\beta-2}V]H^T)$$

Deriving Multiplicative Update Rules

We know that :

$$H^{(i+1)} \leftarrow H^{(i)} \bigotimes \frac{\nabla_H^- \beta_{Div}}{\nabla_H^+ \beta_{Div}}$$

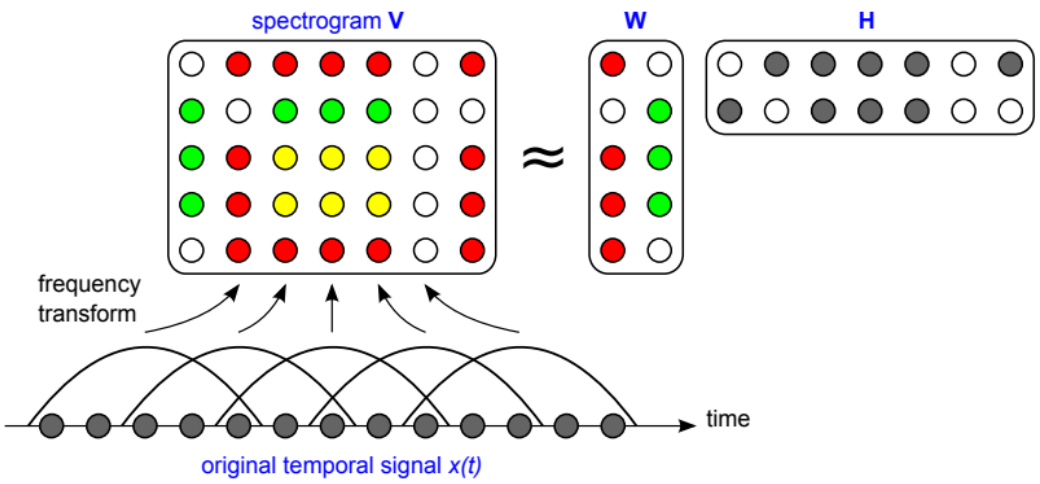
$$W^{(i+1)} \leftarrow W^{(i)} \bigotimes \frac{\nabla_W^- \beta_{Div}}{\nabla_W^+ \beta_{Div}}$$

Using the above-mentioned formulas and the derivatives, we get:

$$H^{(i+1)} \leftarrow H^{(i)} \bigotimes \frac{W^T[(WH)^{\beta-2}V]}{W^T(WH)^{\beta-1}}$$

$$W^{(i+1)} \leftarrow W^{(i)} \bigotimes \frac{[(WH)^{\beta-2}V]H^T}{(WH)^{\beta-1}H^T}$$

Example



Steps for extracting audio sources using NMF

Importing Libraries

```
# Import libraries
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import librosa
import librosa.display
import IPython.display as ipd
```

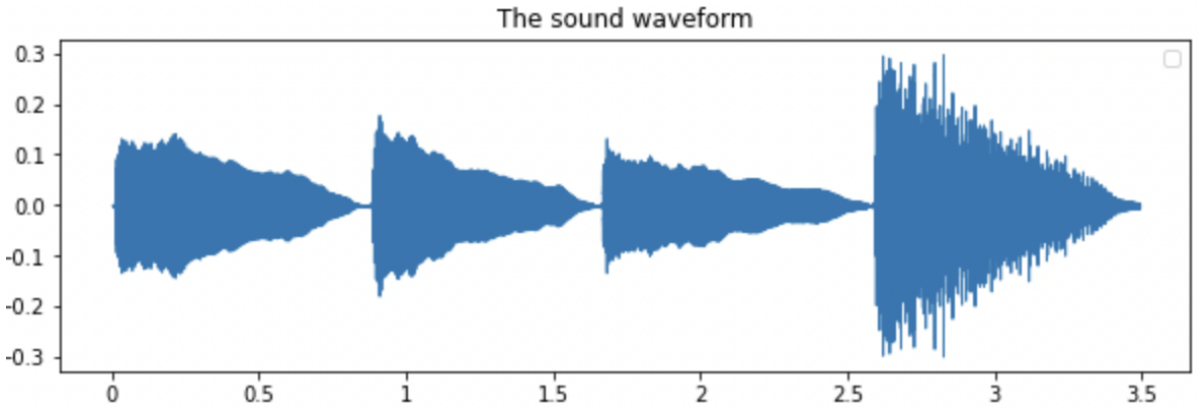
Load and display the audio sound signal

```
# Load audio recording signal
audio_file = 'piano_mix.wav'
sample_rate = 5512
audio_sound, sr = librosa.load(audio_file, sr = sample_rate)

# Display audio
ipd.Audio(audio_sound,rate = sr)

# Plotting the sound's signal waveform
fig, ax = plt.subplots(figsize=(10, 3))
librosa.display.waveshow(audio_sound, sr=sr,
ax=ax,x_axis='time')
ax.set(title='The sound waveform', xlabel='Time [s]')
ax.legend()
```

The audio consists of 3 consecutive different piano notes followed by the 3 notes played together.



Compute Short Term Fourier Transform of the signal

An audio sound is a signal in which its frequency components vary over time. To have a good representation of such signals, we use the Short-Term Fourier Transform (STFT), where its complex-valued coefficients provide the frequency and phase content of local sections of a signal as it evolves over time.

A spectrogram is a visual representation of the changing spectra as a function of time. Each column of a spectrogram represents a time frame of the audio, and each row represents a certain frequency.

The spectrogram corresponds to the squared magnitude of the STFT.

$$spectrogram(t, \omega) = |STFT(t, \omega)|^2$$

We will be using this spectrogram as our data matrix V in the NMF, as it satisfies the non-negativity criterion.

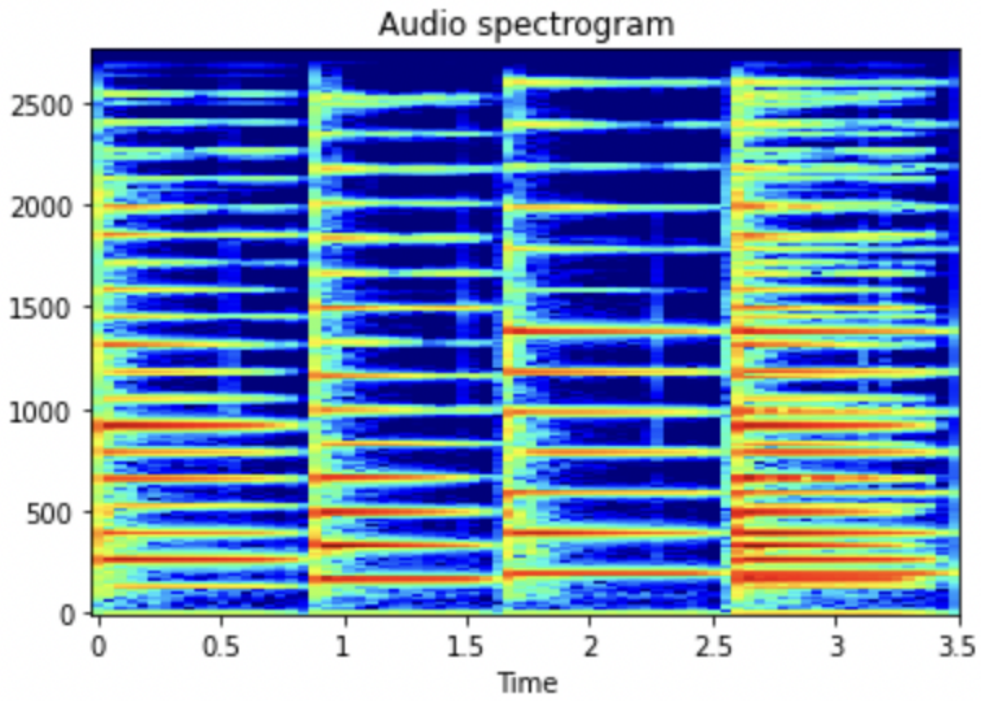
```
# Return the complex Short Term Fourier Transform
sound_stft = librosa.stft(audio_sound, n_fft = 512,
hop_length = 256)

# Magnitude Spectrogram
sound_stft_Magnitude = np.abs(sound_stft)

# Phase spectrogram
sound_stft_Angle = np.angle(sound_stft)

#Plot Spectogram
Spec = librosa.amplitude_to_db(sound_stft_Magnitude, ref =
np.max)
librosa.display.specshow(Spec,y_axis =
```

```
'hz',sr=sr,hop_length=HOP,x_axis ='time',cmap=
matplotlib.cm.jet)
plt.title('Audio spectrogram')
```



Apply the Non-Negative Matrix Factorisation

The algorithm consists of 3 steps:

1. Randomly initialize W and H matrices with non negative values.
2. Update W and H, by considering the other fixed.
3. Repeat 2 until the convergence threshold is hit.

Define a function that computes the β **Divergence**

```
def divergence(V,W,H, beta = 2):

    """
    beta = 2 : Euclidean cost function
    beta = 1 : Kullback-Leibler cost function
    beta = 0 : Itakura-Saito cost function
    """
```

```
    if beta == 0 : return np.sum( V/(W@H) - 
math.log10(V/(W@H)) -1 )

    if beta == 1 : return np.sum( V*math.log10(V/(W@H)) + 
(W@H - V))

    if beta == 2 : return 1/2*np.linalg.norm(W@H-V)
```

Define the main NMF function

```
def NMF(V, S, beta = 2,  threshold = 0.05, MAXITER = 5000):

    """
    inputs :
    --------
        V         : Mixture signal : |TFST|
        S         : The number of sources to extract
        beta      : Beta divergence considered, default=2 
(Euclidean)
        threshold : Stop criterion
        MAXITER   : The number of maximum iterations, 
default=1000

    outputs :
    ---------
        W : dictionary matrix [KxS], W>=0
        H : activation matrix [SxN], H>=0
        cost_function : the optimised cost function over 
iterations

   Algorithm :
   -----------

    1) Randomly initialize W and H matrices
    2) Multiplicative update of W and H
    3) Repeat step (2) until convergence or after MAXITER
    """
```

```
    counter = 0
    cost_function = []
    beta_divergence = 1

    K, N = np.shape(V)

    # Initialisation of W and H matrices : The initialization
is generally random
    W = np.abs(np.random.normal(loc=0, scale = 2.5,
size=(K,S)))
    H = np.abs(np.random.normal(loc=0, scale = 2.5,
size=(S,N)))

    while beta_divergence >= threshold and counter <=
MAXITER:

        # Update of W and H
        H *=
(W.T@(((W@H)**(beta-2))*V))/(W.T@((W@H)**(beta-1)) + 10e-10)
        W *= (((W@H)**(beta-2)*V)@H.T)/((W@H)**(beta-1)@H.T +
10e-10)

        # Compute cost function
        beta_divergence =  divergence(V,W,H, beta = 2)
        cost_function.append( beta_divergence )
        counter += 1

    return W,H, cost_function
```

Running the NMF algorithm on the data matrix

```
V = sound_stft_Magnitude + 1e-10
beta = 2
S = 3

# Applying the NMF function
W, H, cost_function = NMF(V,S,beta = beta, threshold = 0.05,
```

```
MAXITER = 5000)

# Plotting the cost function
plt.figure(figsize=(5,3))
plt.plot(cost_function)
plt.title("Cost Function")
plt.xlabel("Number of iteration")
plt.ylabel(f"Beta Divergence for beta = {beta} ")
```

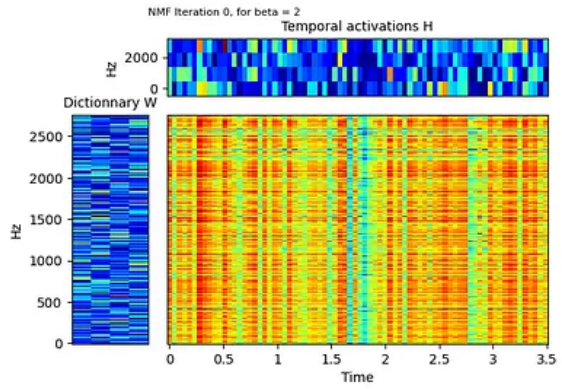
Visualizations

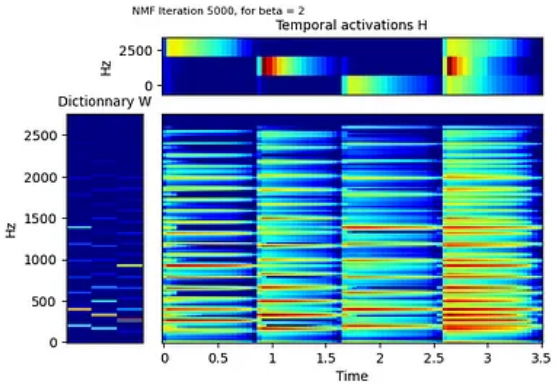
In thefigure, we can see a visual plot of the W, H, and their product WH which represent an approximation of the data matrix V.

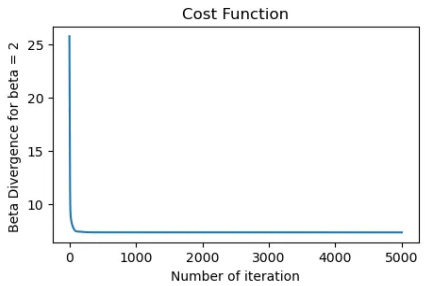
$$V \approx WH = \hat{V}$$

Left : Random initialization of the W and H matrices.

Right : The last iteration.







Now, if we look closely at the activation matrix H, we can see that the algorithm has successfully been able to distinguish the three consecutive notes as different audio sources (it has separated it into 3 sources because we have set S, the common dimension of W and H, to 3).

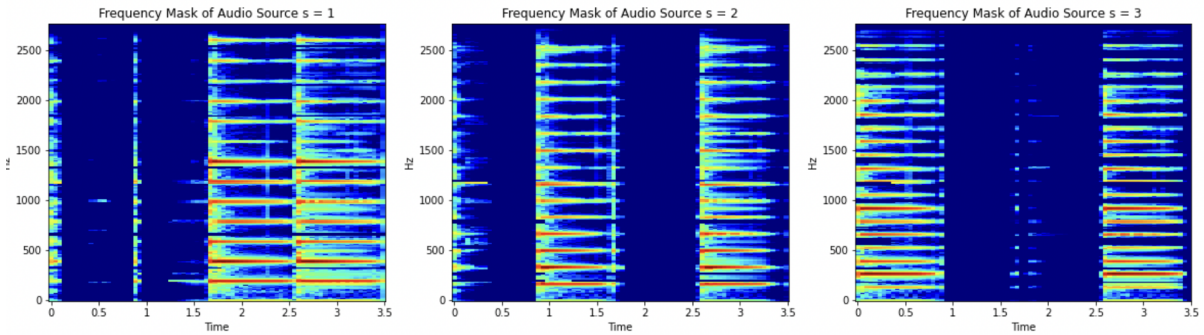
Filtering the different audio sources

Each extracted sound source k, can be calculated by multiplying the k-th column of W and k-th line of H.

$$S_k = W_k H_k$$

```
#After NMF, each audio source S can be expressed as a
frequency mask over time
f, axs = plt.subplots(nrows=1, ncols=S,figsize=(20,5))
filtered_spectrograms = []
for i in range(S):
    axs[i].set_title(f"Frequency Mask of Audio Source s =
{i+1}")
    # Filter each source components
    filtered_spectrogram = W[:,[i]]@H[[i],:]
    # Compute the filtered spectrogram
    D = librosa.amplitude_to_db(filtered_spectrogram, ref =
np.max)
    # Show the filtered spectrogram
    librosa.display.specshow(D,y_axis = 'hz',
sr=sr,hop_length=HOP,x_axis ='time',cmap= matplotlib.cm.jet,
ax = axs[i])

    filtered_spectrograms.append(filtered_spectrogram)
```



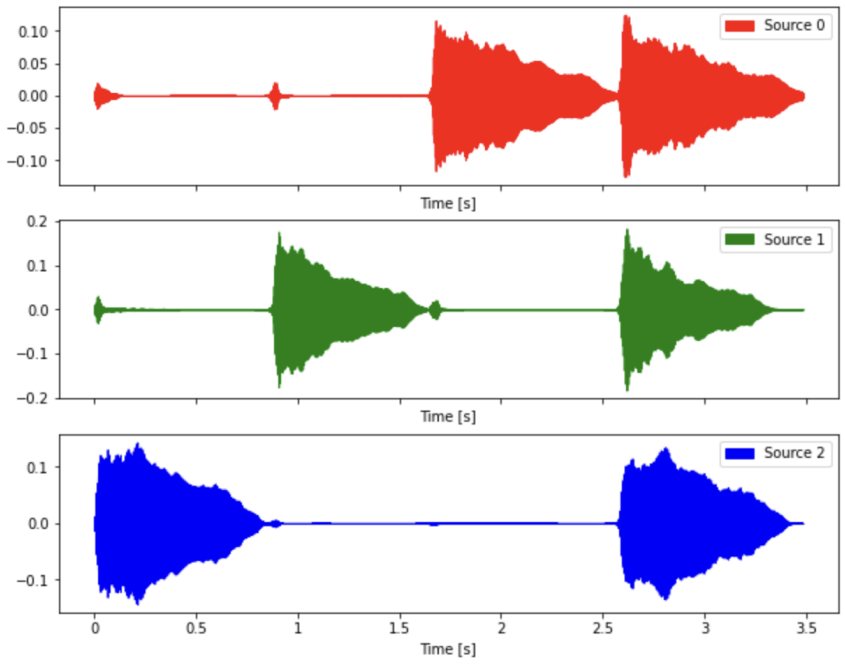
Reconstructing source's audio signals

The phase of the sources is recovered by simply adding the phase of the original mixture audio, which was computed by the STFT, to the filtered sources.

The three audio source signals are then reconstructed with the inverse STFT.

```
reconstructed_sounds = []
for i in range(S):
    reconstruct = filtered_spectrograms[i] *
np.exp(1j*sound_stft_Angle)
    new_sound   = librosa.istft(reconstruct, n_fft = FRAME,
hop_length = HOP)
    reconstructed_sounds.append(new_sound)
```

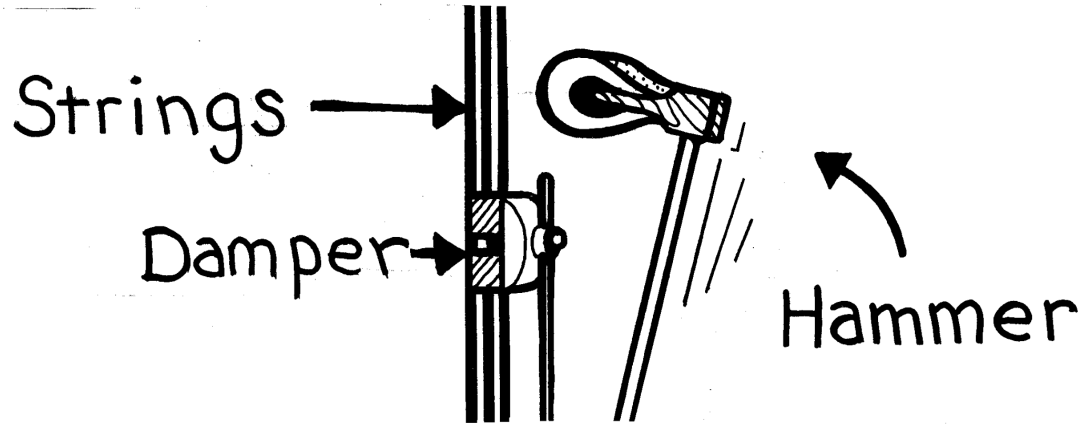
```
# Tracing the waveform
colors = ['r', 'g','b']
fig, ax = plt.subplots(nrows=3, ncols=1, sharex=True,
figsize=(10, 8))
for i in range(S):
    librosa.display.waveshow(reconstructed_sounds[i], sr=sr,
color = colors[i], ax=ax[i],label=f'Source
{i}',x_axis='time')
    ax[i].set(xlabel='Time [s]')
    ax[i].legend()
```

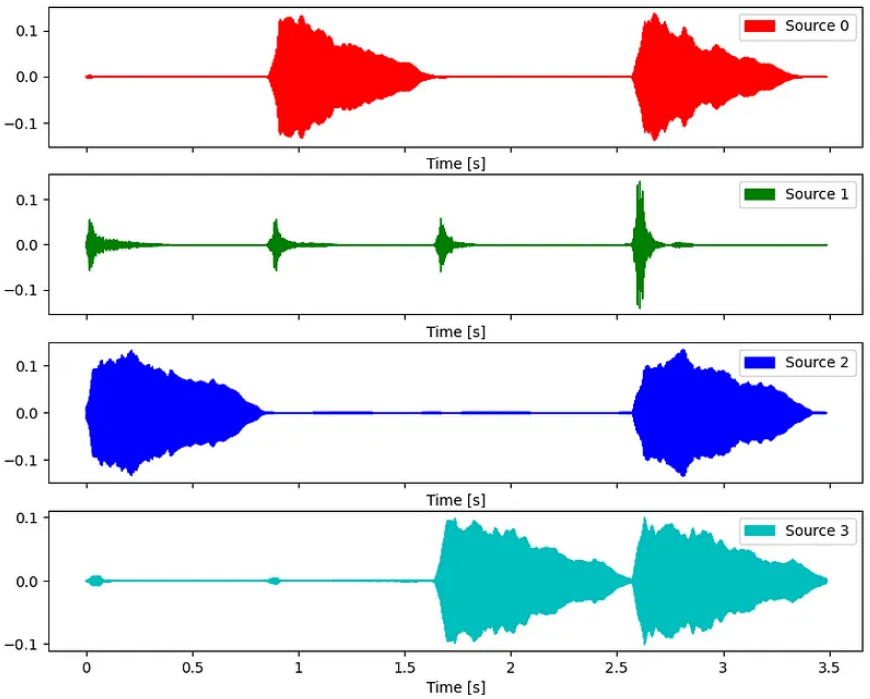


Challenges

One of the challenges in applying NMF to sound source separation is determining the appropriate number of sound sources. NMF can be applied in several different ways to address this issue. One popular method is to use an iterative algorithm that starts with a small number of sound sources, and then increases the number of sound sources until the separation is satisfactory.

In our case, if S, the number of sources to extract is set to S = 4. We are able to extract the sound of the actionning the piano's hammer. (Source 2 , in green below)





Conclusion

Non-negative matrix factorization (NMF) is a powerful sound source separation technique that can extract individual sound sources from a mixture of sounds. NMF is well suited for audio signals as it enforces non-negativity, which is a desirable property for audio signals.

References

1. https://inria.hal.science/hal-01631185/document
2. https://proceedings.neurips.cc/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf