CMRM Homework Assignment No. 1 (HWA1)

Veronica Coccoli – 10706276

Gregorio Secchi – 10680774

a.a. 2023/2024

1 Question 1

The aim of the project is to achieve source separation in layered music tracks. In particular, the goal is to separate each canonical voice (among *basso, tenor, alto, soprano*) used in chorales. The information is already split into three folders:

- **midi**: it contains all midi files for each voice related to each of the four chorales;
- **wav**: it contains all wav files for each voice related to each of the four chorales;
- **mix**: it contains the chorales' wav files, this time with no voice separation.

1.1 Extract annotations

First of all, a function to extract midi notes' information (start, duration, pitch, velocity, label) is implemented. In order to obtain those, a module `pretty_midi` is imported, which comes already with methods allowing to extract all required information from midi files.

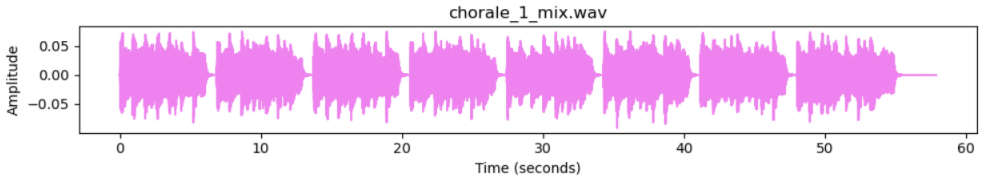


Figura 1: Chorale $n.1$ waveform.

The first chorale to be processed is chorale $n.1$ (its waveform can be observed in above figure 1): the midi instrument is analyzed for each voice among `['soprano', 'alto', 'tenor', 'bass']`, and desired information is retrieved and stored in appropriate vectors. To be more precise, at the end of the `for` loop, one will come up with six vectors: five of them (start, duration, pitch, velocity, label) will be list of lists, having length of 4 (thus, one list of midi information for each voice), in a format that would be like:

```
start = [[start, start, ...], [start, start, ...], [], []]
```

While a peculiar vector *annotations* will contain all the lists defined till now in the form :

```
annotations = [[start, duration, pitch, velocity, label], [], [],...]
```

If *verbose* argument of the defined function is set to `true`, a plot of the piano roll will also be provided (see figure 2)

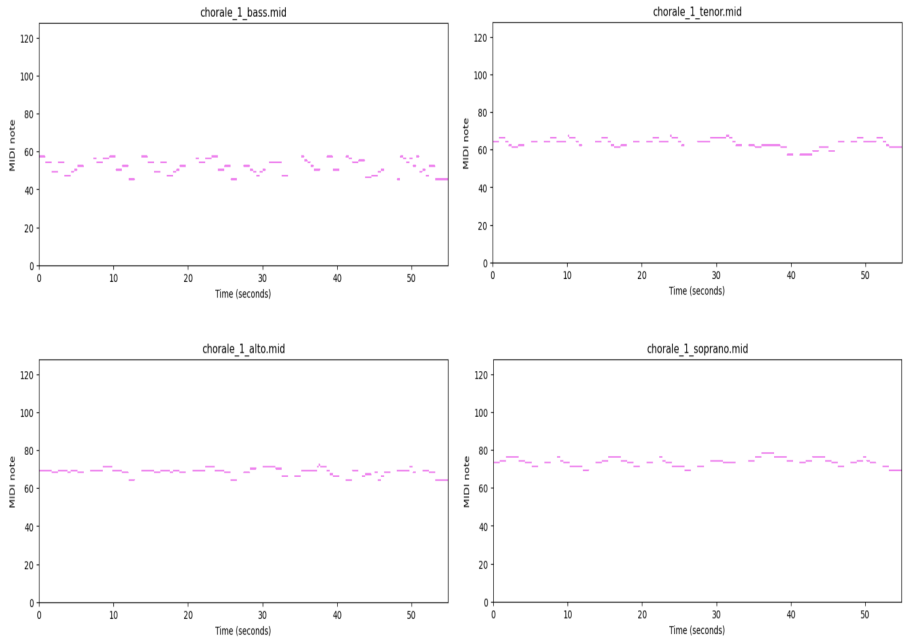


Figura 2: Piano rolls for chorale's $n.1$ voices.

1.2 Create dataframe

Using the data stored in the vector *annotations*, a `pandas.DataFrame` is created and printed (figure 3). It results in being `[238 rows x 5 columns]`, meaning that start, duration, pitch, velocity and label of 238 midi note events have been stored in the dataframe.

2 Question 2

2.1 NMF model

The current project concerns **NMF-Based Audio Decomposition**[1]. Among all source separation tecniques, NMF procedure is pretty popular when audio recording's notewise fashion is at disposal, like in our case. To perform NMF audio decomposition, the following steps have been implemented:

- The STFT[2] of audio signal x is calculated, and the matrix V, which can be thought of as a time–frequency representation consisting of a sequence of spectral vectors, is obtained [1]. Logarithmic compression is later applied in order to balance the values encountered in the range of magnitude spectrogram.

```
X = librosa.stft(x, n_fft=N_fft, hop_length=H_fft)
V = np.log(1 + np.abs(X), dtype = np.float64)
```

[1] Non-Negative Matrix Factorization
[2] Short-Time Fourier Transform

	start	duration	pitch	velocity	label
0	0.0	0.857143	73	90	soprano
1	0.857143	0.857143	74	90	soprano
2	1.714286	0.8571430000000002	76	90	soprano
3	2.571429	0.8571429999999998	76	90	soprano
4	3.428572	0.8571429999999998	74	90	soprano
..	...	...	...	...	...
233	51.0000085	0.4285714999999968	49	90	bass
234	51.42858	0.4285715000000039	50	90	bass
235	51.8571515	0.4285714999999968	47	90	bass
236	52.285723	0.8571430000000007	52	90	bass
237	53.142866	1.7142860000000013	45	90	bass

Figura 3: Dataframe storing MIDI annotations of chorale $n.1$.

The data type of V is set to float64 as it has to be compatible with W and H dtype too.

- Define frequency resolution and frame resolution, i.e.

```
freq_res = Fs / N_fft
frame_res = H_fft / Fs
```

with N_fft being window size and H_fft being hop size.

- Define two variables containing the shapes of V, which is a matrix with K rows (which can be seen as K spectral coefficients) and N columns (N time frames).

```
K = np.shape(V)[0]
N = np.shape(V)[1]
```

- Initialize activation matrix H_init, which represents the events' occurrence over time.

```
tol_note = [0.1, 0.5]
tol_onset = [0.2, 0.1]
tol_pitch = 0.05
H_init, pitch_set, label_pitch = nmf.
    initialize_activation(N, annotations, frame_res,
    tol_note=tol_note, tol_onset=tol_onset)
```

- Initialize template matrix W_init, which contains template vectors (vectors of spectral coefficients).

```
W_init = nmf.initialize_template(K, pitch_set,
    freq_res, tol_pitch = 0.05)
```

- Initialize NMF model, using module `sklearn.decomposition`.

```
model = skl.NMF(n_components=np.shape(H_init)[0],
    solver='mu', init='custom', max_iter=1000)
```

- Perform NMF factorization, i.e. find two non-negative matrices H, W such that their dot product is almost equal to V.

```
W = model.fit_transform(V, W=W_init, H=H_init)
H = model.components_
V_approx = W.dot(H)
```

`V_approx` results being the nonnegative matrix $W * H$ of size KxN, so the spectrogram approximation.

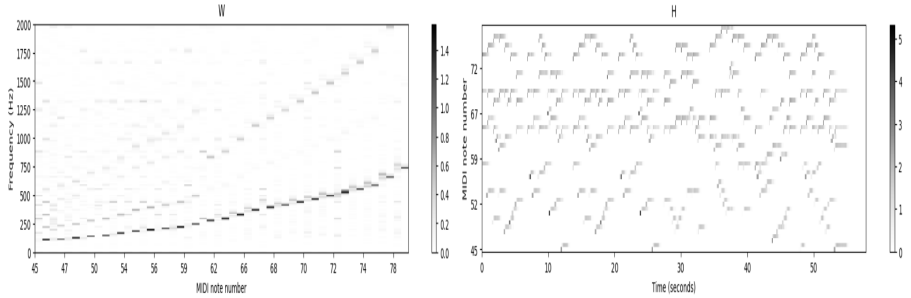
- Print error between `V` and `V_approx`.

```
V_approx_err = np.linalg.norm(V-V_approx, ord=2)
print(f'V approx error: {V_approx_err}')
```

The estimated error is 36.42633676602657.

2.2 Plots

In order to analyze the data, both activation (figure 4b) and template matrix (figure 4a) have been plot.



(a) Template matrix. (b) Activation matrix.

Moreover, figure 5 shows the graph of both original and approximated magnitude spectrogram. As one can notice, the two plots appear very similar. Temporal events, which are discernible by vertical lines, are well defined, while frequency components seem to be quite blurry (yet of course distinguishable). One parameter which could possibly influence the time-frequency representation of V matrix is the length of the window used in calculating the STFT: with a wider window N_fft, more spectral coefficients would be relevant, as frequency resolution `freq_res = Fs / N_fft` would be a smaller number, while on the other side frame resolution would be less quality. Aside from representation issues, different parameter setting for NMF model initialization could result in better approximation among the two matrices. Parameters to be changed include tolerance for note, pitch and onset: those are used in `W_init` and `H_init` construction, therefore they impact on NMF model building (as we use a custom init).

3 Question 3

In this question, some spectral masks are going to be extracted and applied to the activation matrix calculated for each voice in order to extract a first example of sources. A (binary) spectral

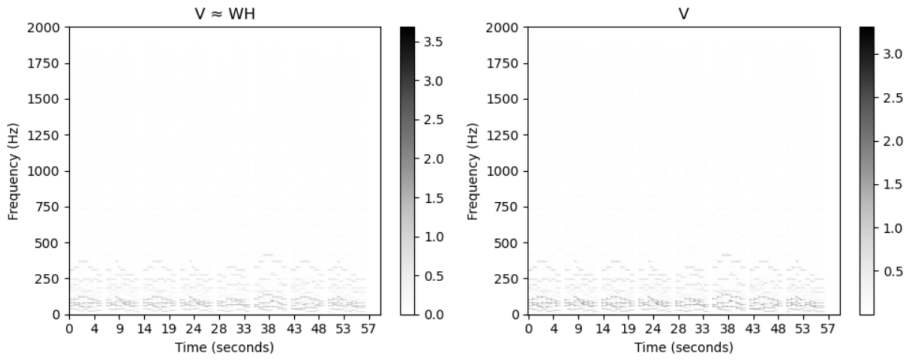


Figura 5: Comparison between approximated and original V matrix.

mask can be applied to activation matrix H to isolate specific note events or components. This spectral mask, when multiplied element-wise with H, keeps only the activated components and discards the rest.

To obtain this, a function `split_annotations` is first defined, which basically separates the MIDI annotations calculated in previous section 1.1 according to their label.

The function is later applied to the annotations of chorale $n.1$. The following code provides three dictionaries having voice labels (`'soprano', 'alto', 'tenor', 'bass'`) as keys, and respectively containing:

- `Hs_init`: all the spectral masks Hs_init for each voice;
- `Hs`: all the activation matrices Hs for each voice, calculated as dot product with the relative and transposed Hs_init spectral mask;
- `Vs`: all the magnitude spectrograms Vs for each voice, calculated as dot product between chorale $n.1$ template matrix W and the relative activation matrix Hs.

```
annotations_split = split_annotations(annotations)
Hs_init = dict(keys=source_names)
Hs = dict(keys=source_names)
Vs = dict(keys=source_names)

for i, name in enumerate(source_names):
    Hs_init[name] = nmf.initialize_activation(N,
        annotations_split[i], frame_res, tol_note=tol_note,
        tol_onset=tol_onset, pitch_set=pitch_set)[0]
    Hs[name] = H * Hs_init[name]
    Vs[name] = W.dot(Hs[name])
```

For each voice, a plot of template matrix W calculated in previous section 2.1, activation matrix Hs and reconstructed spectrogram Vs is provided too. Those are shown in figure 6.

It is indeed interesting to analyze source activation matrices: the pitch range which seems to be more present in each graph is coherent with the voice being processed at the time. For example, in *soprano* activation matrix, MIDI notes around 72 or higher are the most present, while in *alto* voice the most evident pitch component is found around note 70; in *tenor* voice

the heaviest component lies between 59 and 67, while *bass* line captures quite a wide range, spreading from 45 to 59.

All in all, reconstructed sources (as visible from graph of reconstructed magnitude spectrogram WH) are well separated on the basis of their frequency range. The voice which looks more chaotic is the *bass* one, having highlighted not only low frequencies (around or less than 250 Hz), but also plenty of higher ones. That would on the other side make sense, if we consider that bass notes naturally have more harmonics.

4 Question 4

4.1 Soft spectral masking

In order to avoid audible artifacts due to the approximation of magnitude spectrogram, another method (different from the one of previous section 3) is used for masking, which is known as *soft masking*. This basically consists in considering a relative weighting when comparing the magnitudes of spectral coefficients instead of relying on a binary[3] decision [1].

The following code implements soft masking technique applied to each voice of chorale n.1. An explanation of it is provided below.

```
    for i, name in enumerate(source_names):
        Ms[name] = np.divide(W.dot(Hs[name]), W.dot(H) + epsilon)
        stft_masked = Ms[name] * X
        signal[name] = librosa.istft(stft_masked)
```

In each iteration over `source_names`:

- First, a soft mask relative to current-step voice is computed, using the formula $(WH^{voice}) \oslash (WH + \epsilon)$. The smallest epsilon is obtained using `sys.float_info.epsilon`;
- The soft mask is applied through multiplication with STFT;
- A `librosa` function is used for returning to time-domain reconstructed signal.

4.2 Performance rating

Finally, each audio source is plot with its original and reconstructed waveform (a prototype can be seen in figure 7), and an audio player allows to listen to each reconstructed source signal.

Listening to reconstructed sources, we could overall state that source separation was achieved with effort. In particular:

- in the case of *soprano*, the source is finely separated, even though with audible transients (which are actually a common feature to all retrieved sources);
- the *alto* voice is also quite well separated, though having some flaws, as it sometimes captures tenor frequencies;
- viceversa for *tenor* voice, which also presents some sporadic alto frequencies;
- finally, *bass* has a more "choral behaviour": that's something to be expected, as also in section 3, a presence of lots of relevantly-weighted harmonic components had been detected.

[3] Each time–frequency bin is assigned either the value one or the value zero.

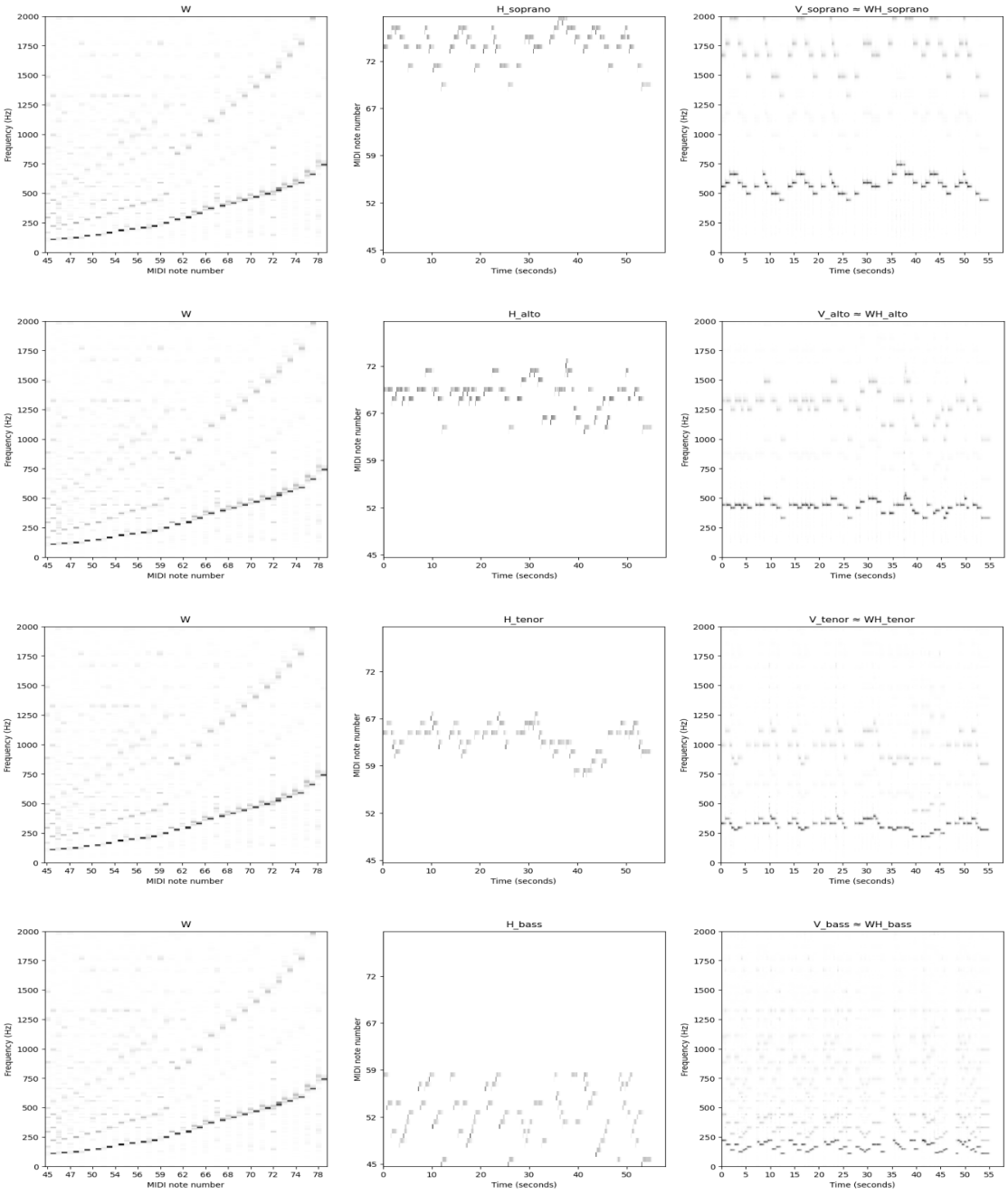
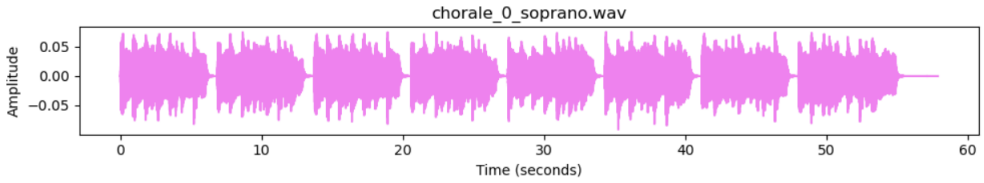
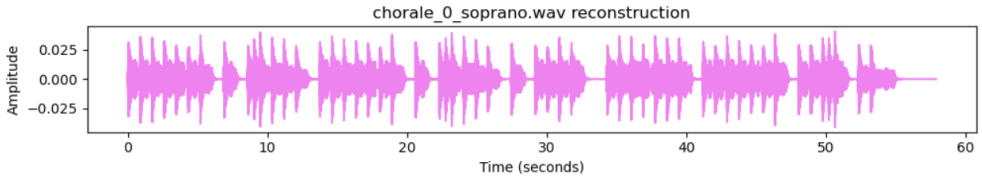


Figura 6: Subplots for each voice of chorale $n.1$.



(a) Original waveform for chorale $n.1$, soprano.



(b) Reconstructed waveform for chorale $n.1$, soprano.

Figura 7

5 Question 5

5.1 Tests

Riferimenti bibliografici

[1] Meinard Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*, volume 5. Springer, 2015.