Second Homework Assignment

of Computer Music Representation and Models

Anna Fusari
Filippo Marri
12th December 2023

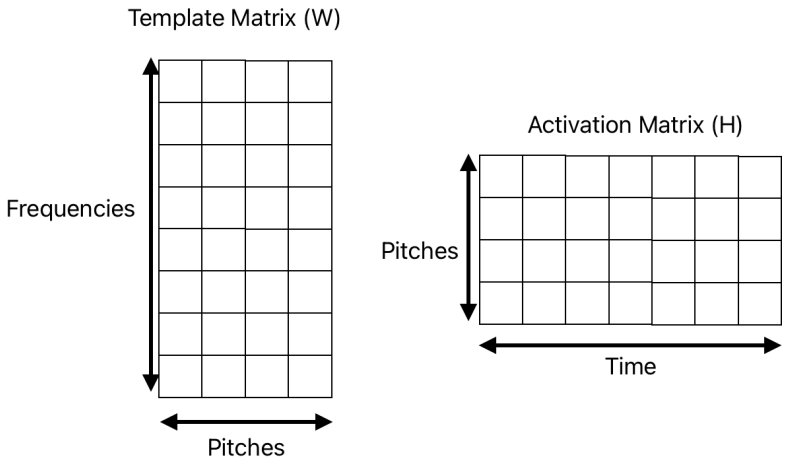
General Description:

The aim of this homework is to create an algorithm able to separate the four voices of a chorale. Nowadays, a task of this kind is solved by resorting to deep-learning architectures. What we want to do is to find a solution to this problem the old-fashioned way adopting the Non-negative Matrix Factorisation (NMF) method.

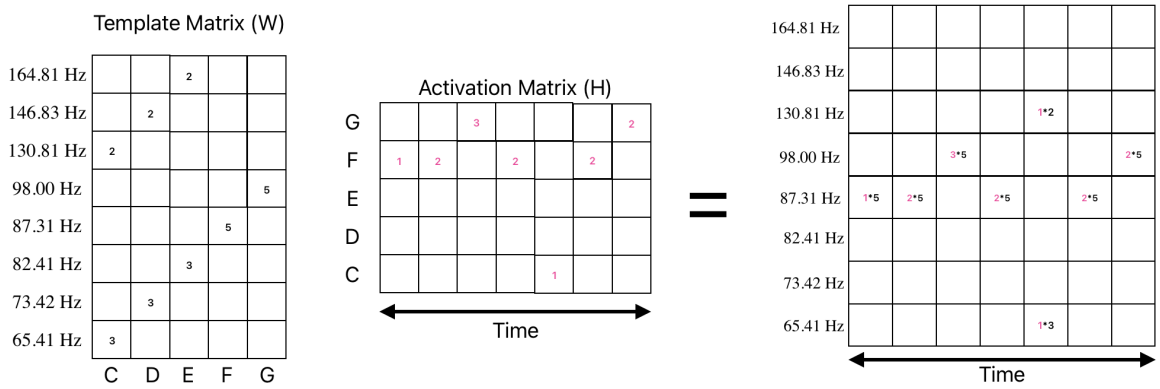
Non-negative Matrix Factorisation:

The NMF has been introduced by Paatero and Tapper, two Finnish researchers, in 1994 under the name of positive matrix factorisation. It has become a popular method with the 1999 article by Lee and Seung. This method is essentially a new way to approximate a certain matrix X by factorising it int two lower-rank matrixes such as $X \simeq WH$.

The **W** matrix is called *template* matrix and it contains the frequency composition of a certain note associated to a certain pitch. The **H** matrix is called *activation matrix* whose cells contain the intensity of the notes that have to be played at a certain time instant. To better understand the system, we can see it as if the activation matrix was the composer who writes the music, while the template matrix is the singer that plays what the composer wrote with its proper timbre. We come back to formality to understand the math behind the NMF method. The structure of the two matrixes is the following:



When we multiply the template matrix by the activation matrix, we obtain another matrix whose elements contains the frequency composition of the sound produced at a certain time instant. Nothing but the STFT. Let us understand it better with a dummy example:



What we will hear during the first time instant will be the note F with an intensity equals to 1 whose timbre is created with just one sinusoid at 87.31 Hz;

then, during the second instant, we will hear the same note F but with a higher intensity;

after that, a G note with an intensity equals to 3 and composed by just one sinusoids that oscillates at 98.00 Hz;

The fourth note will be an F with an intensity equals to 2;

Then a C with an intensity equals to one but with a richer spectra given the fact that this time we have two sinusoids: a first one that oscillates at 65.41 Hz with an amplitude equals to 3 and a second one that oscillates at 130.81 Hz with an amplitude equals to 2;

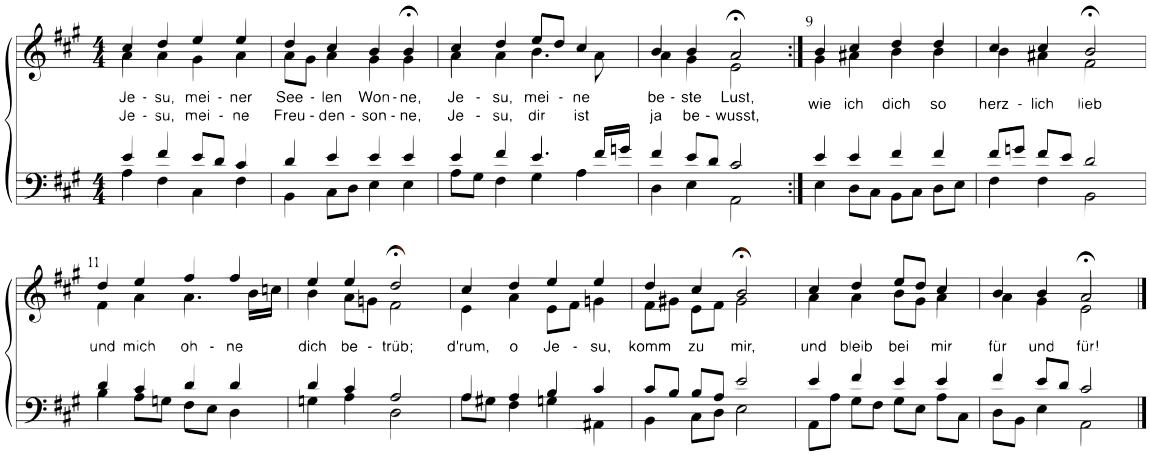
We will have than an F with an intensity equals to 2;

And, at the end, a G with a intensity equals to 2 and composed by just one sinusoid that oscillates at 98.00 Hz;

Chorales:

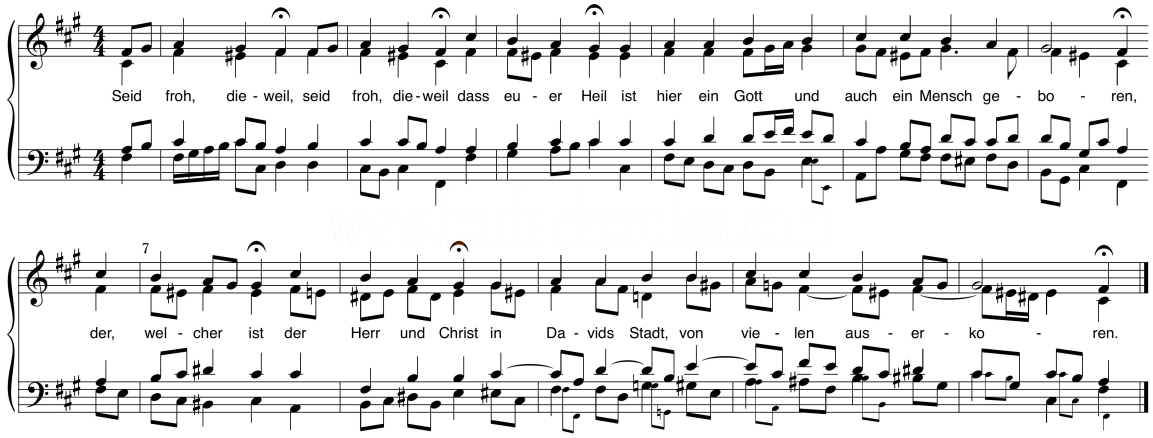
The four chorales we analyse are the following.

The first chorale is **Jesu, meiner Seelen Wonne**, by J.S. Bach (BVW 359). This Choral shares the soprano voice (even thought it is played in other tonalities or with another rhythm) with other Bach's works that are BWVs 55.5, 146.8, 147.6=147.10, 154.3, 244.40, 360. What it changes from a version to another is the harmonisation. This is because, during the baroque period, composers used to write different harmonisation for a same melody that either they or other composers have written. In our case, the tune from which Bach drew inspiration is Werde munter, mein Gemüthe by Johann Schop, a German composer who lived during the first part of the XVI century. As regards the lyrics, the chorale has actually survived without words. The text that is proposed in the sheet music below is the one provided by the editors of the Bach Gesellschaft Ausgabe (BGA) based on the verses by Martin Jahn.



Jesu, meiner Seelen Wonne,	*Jesus, delight of my soul,*
Jesu, meine beste Lust,	*Jesus, my best pleasure,*
Jesu, meine Freudensonne,	*Jesus, my sun of joy,*
Jesu, dir ist ja bewußt,	*Jesus, it is well known to you*
wie ich dich so herzlich liebe	*how I love you from my heart*
und mich ohne dich betrübe.	*and am distressed without you.*
Drum o Jesu komm zu mir	*Therefore O Jesus come to me*
und bleib bei mir für und für!	*and stay with me forever and ever.*

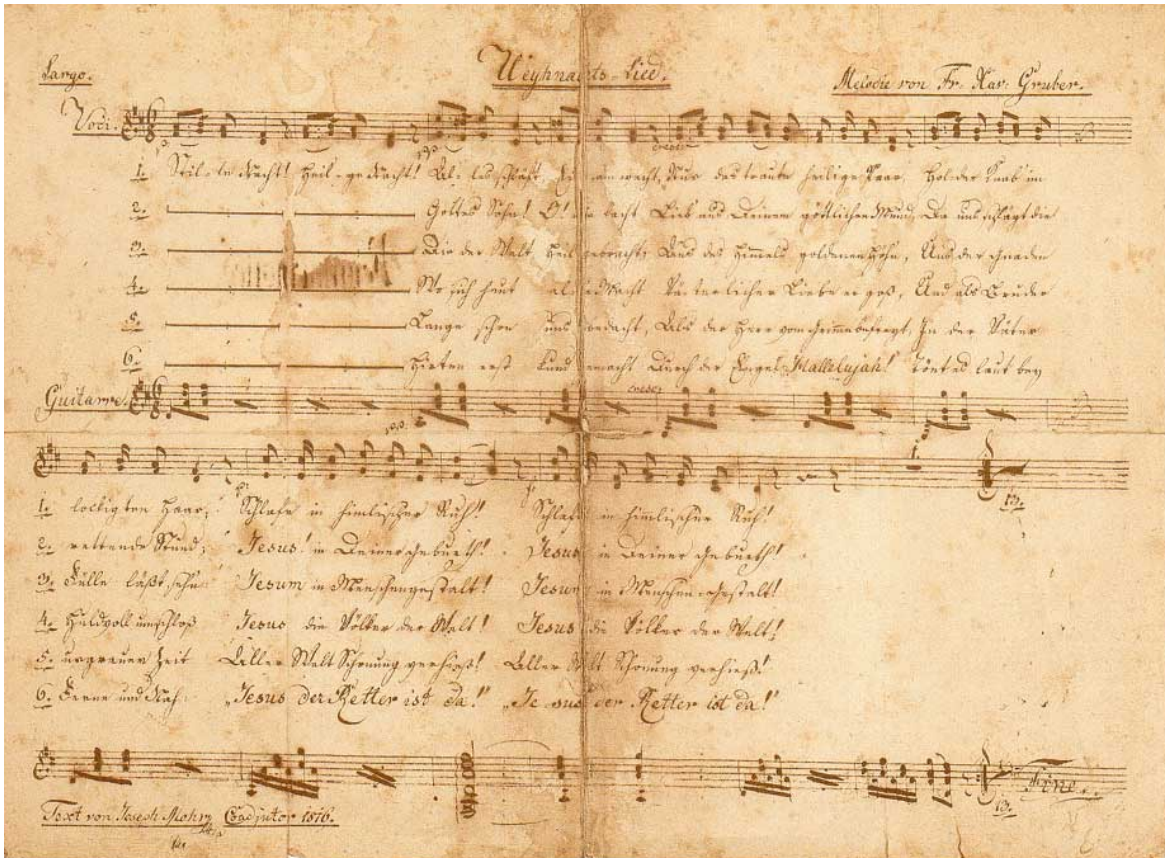
The second chorale is **Seid froh dieweil** also by J.S. Bach (BVW 248.35). It is included in the third part of the *Oratorium tempore Nativitatis Christi (BWV 248),* written for the Christmas season of 1734/1735 when he was the Thomaskantor of Leipzig's Thomaskirche. Its first execution was held on 27th December 1734 during the religious function in which The adoration of the Shepherds it is meditated. Also in this case, Bach harmonised a pre-existent melody taken from an anonymous manuscript dated 1589. The lyrics are taken from the fourth stanza of Laßt Furcht und Pein written by Christoph Runge in 1653.



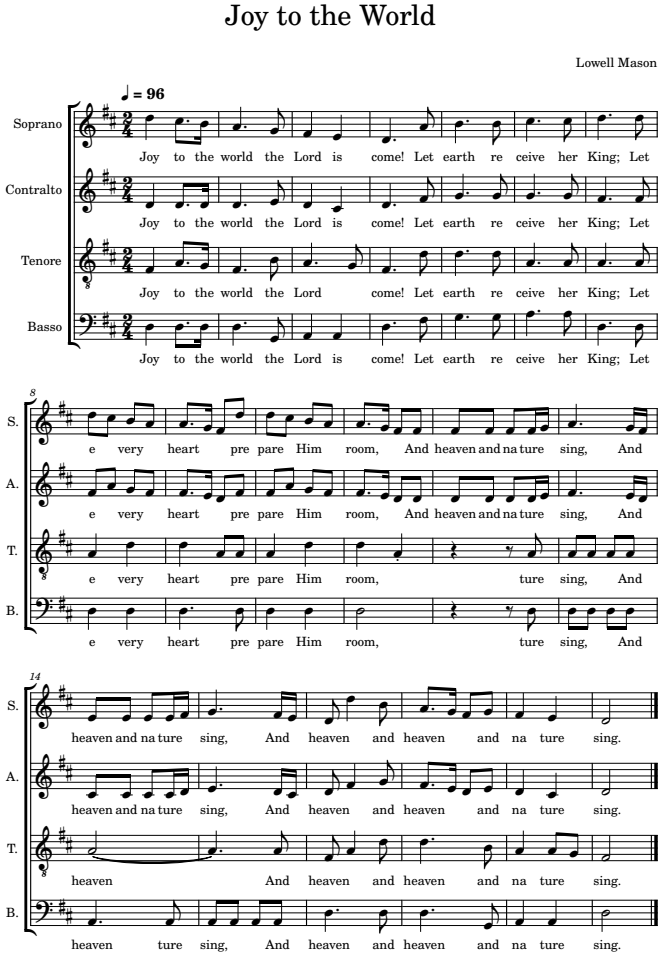
Seid froh dieweil
nun euer Heil
ist heut' ein Gott und auch ein Mensch geboren.
Der, welcher ist
der Herr und Christ,
in Davids Stadt von vielen auserkoren.

Meanwhile, be happy
for your salvation.
It is born here a God and also a person,
He, who is
the Lord and Christ
in David's city, chosen out of many.

The third chorale is a SATB transcription of **Stille Nacht,** a traditional Christmas Song composed in 1818 by F. X. Gruber, schoolteacher and musician, to the lyrics by J. F. Mohr, the young priest of the church of Oberndorf bei Salzburg. During the difficult period of the Napoleonic Wars aftermath, it seems that the young priest walked more than 3 km in the cold of the Christmas Eve of 1818 to ask his friend Gruber, to set his poem to music. In a few hours, Gruber wrote the two melodies and a guitar accompaniment that they played together the next day: Gruber sang the Basso part and Mohr sang the tenor one and played the guitar. It is not clear why it has been written for guitar: someone says because the organ had been destroyed by a flood some days prior and someone says simply because it was Mohr's favourite instrument. After a few years, this tender and delicate song had already become one of the most famous Christmas pieces.



The last chorale is a transcription of **Joy to the world.** The lyrics of this piece were written by Isaac Watts in 1719 inspired by the Psalm 98, while the music was written by the American composer Lowell Mason in 1848. Mason was already well-known at that time given the fact he had composed another famous hymn “Nearer My God to Thee”, piece that had become more sadly famous when, 70 years later, was played by the Titanic orchestra to calm people during the tragic sink. Anyway, on the contrary Joy to the World is characterised by feelings of hope and joy: despite what is seems, the piece does not talk about Christmas, but about “Parusìa” that is the moment when, according to Christian’s view, the Lord will come back at the end of the times. The melody was composed taking some part from Handel’s Messiah, an oratory written in 1741 also a very famous composition due to the well-known Hallelujah. We report not the original version but the one used for this homework.



Joy to the world! the Lord is come;
Let earth receive her King;
Let every heart prepare him room,
And heaven and nature sing,
And heaven and nature sing,
And heaven, and heaven, and nature sing.

Joy to the world! the Saviour reigns;
Let men their songs employ;
While fields and floods, rocks, hills, and plains
Repeat the sounding joy,
Repeat the sounding joy,
Repeat, repeat the sounding joy.

No more let sins and sorrows grow,
Nor thorns infest the ground;
He comes to make His blessings flow
Far as the curse is found,
Far as the curse is found,
Far as, far as, the curse is found.

He rules the world with truth and grace,
And makes the nations prove
The glories of His righteousness,
And wonders of His love,
And wonders of His love,
And wonders, wonders, of His love.

Code analysis:

First of all we import all the libraries, the objects and the function that we need:

- **os:** this library allows us to work with operation of the operative system. We will use it to load the songs.
- **libROSA:** it allows us to work with audio files.
- **pretty_midi:** it is a library that contains functions and classes for handling MIDI data.
- **numpy:** renamed **np** for practical reasons, it allows us to work with arrays.
- **matplotlib.pyplot:** renamed **plt**, is a procedural interface of the library matplotlib that allows us to plot graphs. The library is modelled closely after matLAB. Therefore, the syntax of the majority of plotting commands is similar matLAB's ones.
- **pandas:** as **pd**, a great library to work with tabular data.
- All the functions from the file **nmf_utils** where we will find some interesting function for managing the matrixes of the NMF.
- From the object **decomposition** of **sklearn**, we will import the function **NMF.** The object contains several functions to extract specific features from a large set of data. In our case, we will use the one that allows us to obtain the NMF of the chorales' spectrums.
- The **display** module from **IPython**: a library expressively created to have an interactive shell on Jupyter with didactical aim. The display module just provides a public API for display tools; in our case, the audio tool **ipd.Audio** that will be used to hear the tracks. We will name the module **pd** for simplicity.
- The **mir_eval** library, dubbed **me**, where we can find interesting tools to evaluate Music Information Retrieval systems.

Question One:

First of all, we create some variable whose concatenation corresponds to the piece's storage path. For this question, we are going to work with **Jesu, meiner Seelen Wonne** (BWV 359) by Bach.

Then, we create a list with the name of the four voices of a SATB chorale: soprano, alto, tenor, bass.

We import the audio file correspondent to the path we have specified in the variables above and we create a variable `chorale1` containing the floating point time series version of the chorale we have imported. We remind that the amplitude is usually measured as a function of the change in pressure around the microphone or receiver device that originally picked up the audio; this is why we have chosen Pa as unity of measure even thought our samples are synthesised sounds. We sample the audio file with the librosa default sampling frequency $F_s = 22050\ Hz$.

At this point, we create a vector containing all the time instants related to every sample of the audio file by listing the samples with the function `np.arange` and dividing to the sampling rate.

Then, we define a general plotting function and we call it to see if everything works well. This function takes as input five parameters: the elements on the y axis, the elements on the x axis, the plot's title, the x-axis label and the y-axis label. In this way, the function is written in a way as general as possible.

We can compare the waveform we have obtained with what we hear as a check: the chorale is written as a sequence of musical phrases interspersed with pauses, a structure that can recall the Gregorian chant one perfectly represented by the shape of the waveform. Therefore, we can suppose to be on the right way.

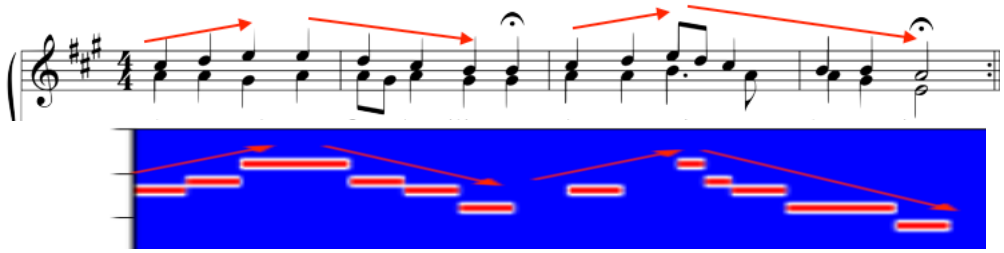
After that, we write a function to extract the annotations contained in the midi file. But, what are these annotations? They are nothing but information regarding the piano roll of the piece. In order are:

- Start: a list that contains the starting sample of each note;
- Duration: a list that contains the duration of each note;
- Pitch: a list that contains the pitch of each note;
- Velocity: a list that contains the note velocities. Pay attention to the false friend. With velocity we do not mean anything that has to do with speed but with intensity. So, this list contains the intensity of each note of the chorale.
- Label: a list that contains the name of the voice analysed. It can be "soprano", "alto", "tenor" or "bass";
- Annotations: it contains the concatenation of the elements of the previous lists organised as a list of lists. For example, the first element of annotation is a list composed by five elements that respectively are the start, the duration, the pitch, the velocity and the label of the first note of the soprano part.

So, we initialise the six lists and then we create a loop made up by three nested for loops. The first one will scan the list "source_names" to select the voice we want to analyse. We import the midi track related to that voice, first thing. The second for loop scans the instrument object that allows us to hold event information for a single instrument. Given the fact that we provide the voices separately we could have avoided this line but we left it in order to maintain the function as general as possible. The last for loop scans the notes of the music line. We are finally in the core of our nested for loop where we append the information related to the note we have selected to the relative list. After that, there is the code for plotting the piano roll. If the verbose parameter is set to true we plot the piano roll with the function `get_piano_roll` contained in the midi_data object. We crop the y-axis

between the pitch values 40 and 80 in order to highlight the pitch section where the notes of the four lines are present.

We are ready now to extract annotations for the first chorale. To better visualise the lists, we organise them into a data frame table by using the function DataFrame from the panda library. Even in this case, we can double check our results: the easiest way is to follow the voices trend in the piano roll and compare them with the sheet music. If we want to be more precise, we can evaluate mathematically if the pitches and the notes actually correspond. We report an example, following the easiest way, for the soprano part.



As we can see the piano roll trend follows very accurately the sheet music. Furthermore, we can notice that, in the midi file, there are some missing notes respect to the sheet music: we will have to keep this in mind when we will discuss the results. One last thing we can appreciate from the data frame is that we can understand just by watching the velocities datas that the chorale is played with synthesisers: in fact, it is almost impossible (and also inhuman) to obtain the same intensity for each note if the piece is singed by a real voice.

Question Two:

We are now ready to start with the proper NMF.

First of all, we define the parameters we need to evaluate the STFT: these are **`N_fft`** that stands for the number of samples on which the STFT is evaluated and the hop size **`H_fft`** so the step size of which the window of the STFT has to be shifted along the signal. We evaluate the STFT by means of `librosa.stft`, the librosa function that returns the Short Time Fourier Transform of the signal put as input. We have explicitly specified the type of window in the function parameters even though we have chosen the default one. This is because we want to emphasise that, in order to avoid artefacts as much as possible, it is better to use an Hann window instead of a simple rectangular impulse when we select the signal portion on which evaluate the STFT. After that, we save the logarithmic compression of the magnitude of the STFT in a variable called **`V`** in order to get a better representation of the spectra. In fact, seeing that the values of a spectrogram possess a large dynamic range, small, but still relevant values may be dominated or obscured by large values if we do not apply a logarithmic compression. We remind that the logarithmic compression of a magnitude is evaluated as it follows:

$$V = \log_{10}\left(1 + \gamma \cdot \left|STFT\,(x)\right|\right)$$

where, for our specific case, $\gamma = 1$. We specify the data type `float64` because we need such data type to evaluate the NMF matrixes.

We define two variables in which we save the frequency resolution and the frame resolution. The former $\left(\frac{F_s}{N_{fft}}\right)$ is the frequency step size between one frequency sample and the other and the latter $\left(\frac{H_{fft}}{F_s}\right)$ is the distance between one time sample and the other inside the window that selects the portion on which to evaluate the DFTF implementation (fft).

We store now the shapes of the spectrogram **V** into two variables. They will come in handy to create the two NFT matrixes since:

- **V_shape0** will contain the shape zero of the spectrogram where there is stored the number of frequencies K in which the signal is decomposed by the stft. **V_shape0** = 2049
- **V_shape1** will contain the first shape of the spectrogram where there is stored the number of time frames N so, simply speaking, the stft time length. **V_shape1** = 1247

We initialise the two matrices of the NMF method by means of two functions from the library `nmf_utils`. We will use `initialize_activation` for the activation matrix and `initialize_template` for the template matrix as their names suggest. Let us dwell on this two functions for a while to better understand how they works.

`initialize_activation:`

The function takes as input six arguments: the length of the STFT that will become the first dimension ("x-axis") of the activation matrix; the annotation list we defined in the first point; the frame resolution; a couple of data that specifies the tolerance the algorithm has to take into account to distinguish two different notes; a couple of data that specifies when we can consider an onset as the starting point of a new note; the `pitch_set` that offers the possibly to work with a predefined pitch set instead of creating a new one with the function. Thus, after some initialisation steps, it is defined the activation matrix that has a number of row (its zero dimension) equals to two times the length of the pitch set, and a number of columns (its first dimension) equals to the length of the STFT as we said. If any pitch set has not been specified in function's arguments, a new pitch set is created. It goes from zero to the highest pitch contained in the pitch section of the

annotation list. Then, there is a for loop that sets equal to one the cells where the pitch indicated by the cell-row is present in the time instant indicated by the cell-column. From there comes the name “activation”: it means that a certain pitch is “activated” at a certain time instant when there is a one in the cell that represents the note that has to be played in that time instant. The function returns the activation matrix, the pitch set and the label pitch.

`intialize_template:`

This second function takes as input four arguments: the number of frequencies in which the signal is decomposed; the `pitch_set` taken from the output of the previous function; the frequency resolution and a parameter that specifies the pitch tolerance that is the minimum difference in pitch that it is recognised as a change of pitch. The function initialise a matrix that has a number of row (its zero dimension) equals to the number of frequencies K, and a number of columns (its first dimension) equals to the two times the length of the pitch set. A for loop fill the matrix with columns that comes from the return of the function `template_pitch`. This function returns the spectral template for a given pitch so, simply speaking, the frequency recipe that composes the timbre of our voices for each pitch. As a result, the function returns the template matrix.

We can now instantiate the NMF model creating the object `NMF_model` with the function `NMF` contained in the library `sklearn.decomposition`. This function takes as input the length of the `pitch_set` that represents the number of components; a parameter called `solver` set to `mu` to use a multiplicative update solver instead of a a coordinate descent one; a parameter called `init` set equals to `custom` to specify that we want to provide both the activation and template matrix by ourself; a last parameter, `max_iter`, that allows us to specify a maximum number of iteration before timing out. As we can imagine, the higher is this parameter the more

accurate will be the result, but a the same time, the slower will be the algorithm.

After that, we perform the actual NMF factorisation by calling the method `fit_transform` that take as input the logarithmic compressed version of the STFT (**`V`**), the activation matrix (**`H_init`**) and the template matrix (**`W_init`**). This function firstly fits the NMF model for the data **`V`** and then it transforms it according to the **`W_init`** and **`H_init`** matrixes that we have provided. It also returns the transformed data that we save in the **`template_matrix`** array. We save also the activation matrix in an array of the same name by setting it equals to the attribute `.components_`. We do a little check on matrixes shapes before going on. We obtain the following results:

```
The shape of the W (template) matrix is (2049, 50)
The shape of the H (activation) matrix is (50, 1247)
```

The result is perfectly coherence with the theoretical discussion since the zero dimension of the template matrix is equal to the number of frequencies in which the signal is decomposed (that we called **`V_shape0`**) and the first dimension of the activation matrix is equal to the number of frames (**`V_shape1`**). Furthermore, the template matrix's first dimension is equal to the activation matrix's zero dimension. A good thing for two reasons: because the two matrixes have to be multipliable and because 50 is exactly two times the dimension of the pitch set that it is equal to 25, as we can see printing the pitch_set dimension.

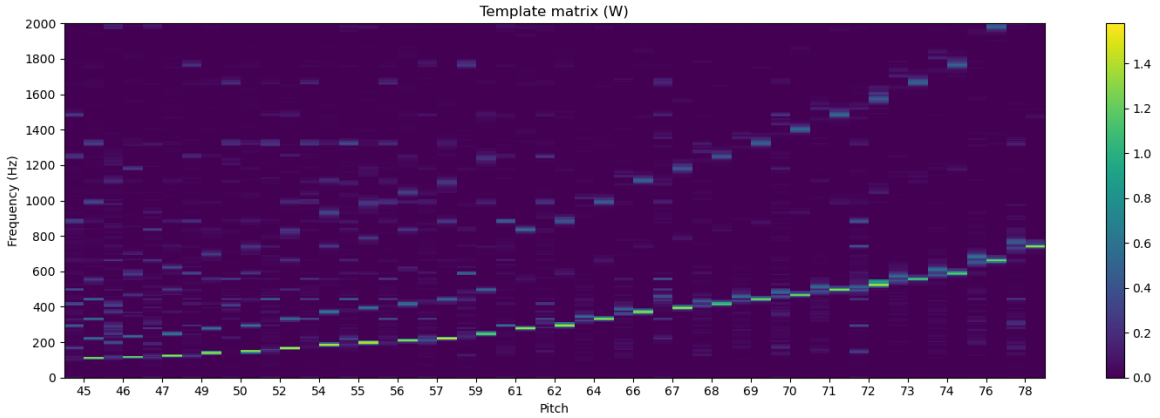
We find now the approximated version of **`V`** by algebraically multiplying the **`template matrix`** to the **`activation matrix`**. We call the result **`V_approx`**. According to tradition, we perform another check by comparing the shape of the matrixes **`V`** and **`V_approx`** that perfectly match.

After that, we compute the 2-norm of the error evaluated as the difference element-wise between **`V`** and **`V_approx.`**

```
The 2-norm of the error is equal to 36.42633676602655
```

We define now a function that allows us to plot all the matrixes in different configurations: if we have just one matrix, the matrix alone; if we have two matrixes in a subplot made up by the two matrixes; if we have 2 or more matrixes in subplots made up by three matrixes.

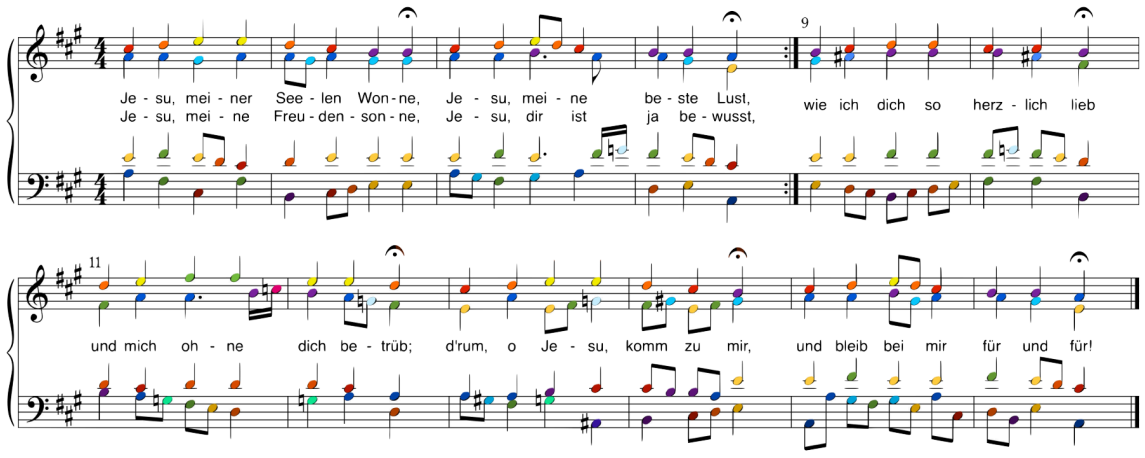
Let us analyse the **`template matrix`**:



as we said before, it represents the Fourier recipe to create the timbre we need for our note: each column represents the frequency decomposition (the spectra) of the note with the pitch selected by the row. We can notice that the frequency increases as we go on along the pitch axis. This behaviour is completely predictable given that there is an exponential relation between pitch and frequency. Just at first glance, we can notice this type of trend. Furthermore, we can observe how for each note the fundamental frequency, the green/yellow one, is stronger than the frequencies above. This is the typical behaviour of the spectra of a sound. Another interesting aspect is that the notes used for alto soprano and alto parts identified by a pitch greater than 61 (C#4) have a poorer spectra than the notes used for tenor and bass parts. This curiosity could explain why the masculines timbers sound darker than the feminines one. A little check before going on: we verify whether the pitch-set contains exactly all the notes that are presents in the chorale. In the

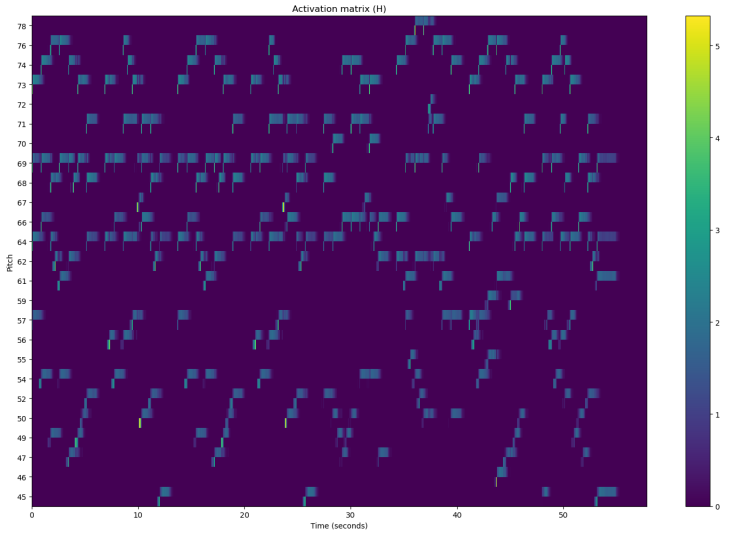
following table are reported the correspondence between fundamental frequency (first number in the cell), pitch (second number in the cell in brackets), octave (column) and name of the note (row). The colours represent the correspondence between the note of the chorale and the elements on the table.

Octave / Note	-1	0	1	2	3	4	5	6	7	8	9	10
C	8.175799 (0)	16.35160 (12)	32.70320 (24)	65.40639 (36)	130.8128 (48)	261.6256 (60)	523.2511 (72)	1046.502 (84)	2093.005 (96)	4186.009 (108)	8372.018 (120)	16744.04
C♯/D♭	8.661957 (1)	17.32391 (13)	34.64783 (25)	69.29566 (37)	138.5913 (49)	277.1826 (61)	554.3653 (73)	1108.731 (85)	2217.461 (97)	4434.922 (109)	8869.844 (121)	17739.69
D	9.177024 (2)	18.35405 (14)	36.70810 (26)	73.41619 (38)	146.8324 (50)	293.6648 (62)	587.3295 (74)	1174.659 (86)	2349.318 (98)	4698.636 (110)	9397.273 (122)	18794.55
E♭/D♯	9.722718 (3)	19.44544 (15)	38.89087 (27)	77.78175 (39)	155.5635 (51)	311.1270 (63)	622.2540 (75)	1244.508 (87)	2489.016 (99)	4978.032 (111)	9956.063 (123)	19912.13
E	10.30086 (4)	20.60172 (16)	41.20344 (28)	82.40689 (40)	164.8138 (52)	329.6276 (64)	659.2551 (76)	1318.510 (88)	2637.020 (100)	5274.041 (112)	10548.08 (124)	21096.16
F	10.91338 (5)	21.82676 (17)	43.65353 (29)	87.30706 (41)	174.6141 (53)	349.2282 (65)	698.4565 (77)	1396.913 (89)	2793.826 (101)	5587.652 (113)	11175.30 (125)	22350.61
F♯/G♭	11.56233 (6)	23.12465 (18)	46.24930 (30)	92.49861 (42)	184.9972 (54)	369.9944 (66)	739.9888 (78)	1479.978 (90)	2959.955 (102)	5919.911 (114)	11839.82 (126)	23679.64
G	12.24986 (7)	24.49971 (19)	48.99943 (31)	97.99886 (43)	195.9977 (55)	391.9954 (67)	783.9909 (79)	1567.982 (91)	3135.963 (103)	6271.927 (115)	12543.85 (127)	25087.71
A♭/G♯	12.97827 (8)	25.95654 (20)	51.91309 (32)	103.8262 (44)	207.6523 (56)	415.3047 (68)	830.6094 (80)	1661.219 (92)	3322.438 (104)	6644.875 (116)	13289.75	26579.50
A	13.75000 (9)	27.50000 (21)	55.00000 (33)	110.0000 (45)	220.0000 (57)	440.000 0 (69)	880.000 0 (81)	1760.000 (93)	3520.000 (105)	7040.000 (117)	14080.00	28160.00
B♭/A♯	14.56762 (10)	29.13524 (22)	58.27047 (34)	116.5409 (46)	233.0819 (58)	466.1638 (70)	932.3275 (82)	1864.655 (94)	3729.310 (106)	7458.620 (118)	14917.24	29834.48
B	15.43385 (11)	30.86771 (23)	61.73541 (35)	123.4708 (47)	246.9417 (59)	493.8833 (71)	987.7666 (83)	1975.533 (95)	3951.066 (107)	7902.133 (119)	15804.27	31608.53

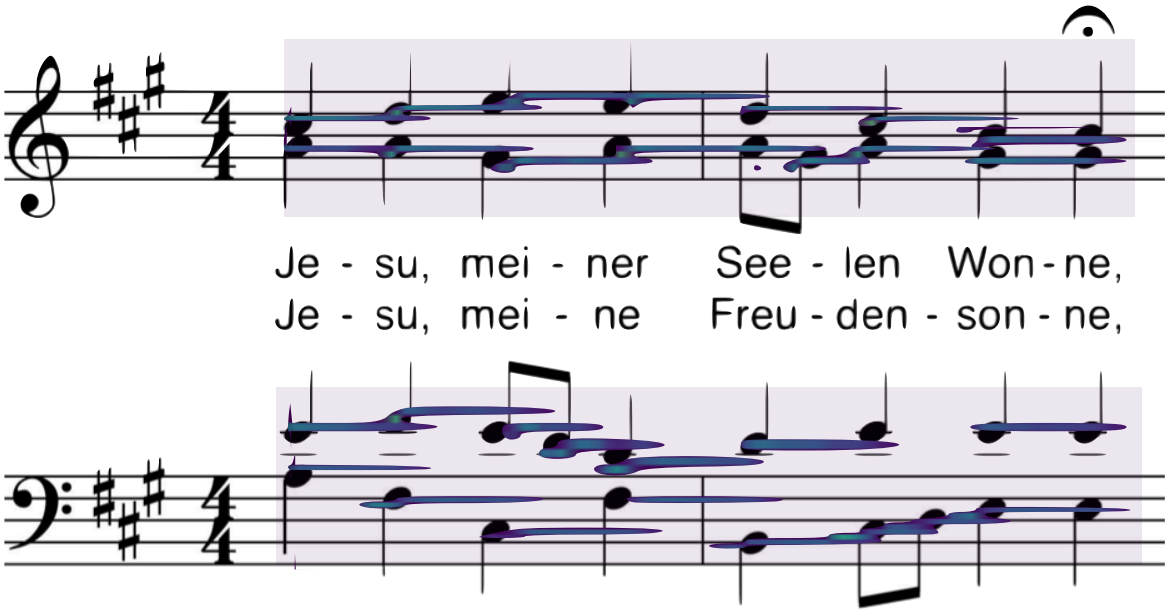


As we can see, they are exactly the pitches contained in the pitch-set.

Let us analyse the **`activation matrix`**:

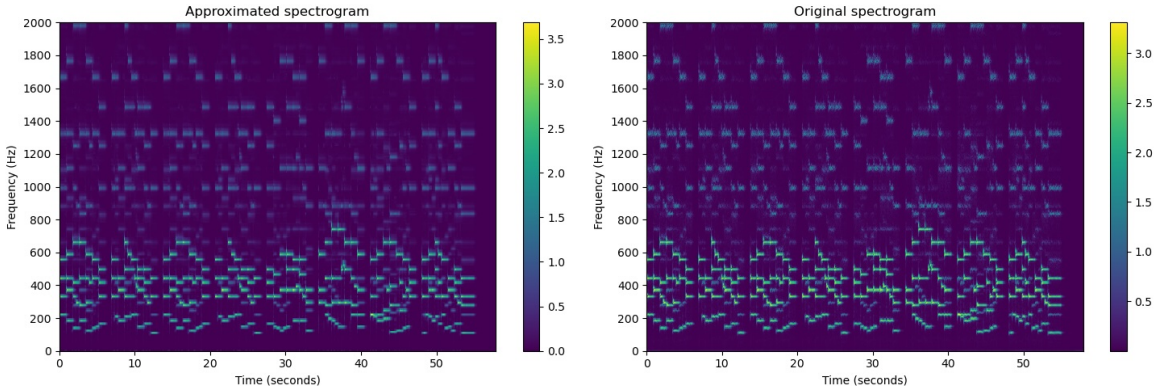


Even in this case we can picture what we have explained before. The activation matrix is a sort of piano roll of all the voices put together. In fact, it shows how many pitches have to be "activated" in a certain time instant. as further proof of the fact, we can notice that there are always not more than four horizontal lines for every time instant: exactly the number of voices of the classic chorale. We can proceed with a little trick, if we are not convinced yet. We superpose the first seven seconds of the activation matrix with the first phrase from the sheet music. As we can see, the trends perfectly match!



We can also notice the absence of some notes in the activation matrix. For instance the E in the second movement of the second bar in the tenor voice.

As last step, we plot the two spectrograms.



At first glance, they would look identical if it was not for a little amplification. We can spot it from the different distribution of the numbers along the colour bar. Anyway, if we look better, the colours in the original spectrogram look closer to yellow than the ones in the approximated spectrogram so there could not be any differences in intensity. As regards the artefacts introduced by the algorithm, the best way to pinpoint them is superimposing the two graphs and changing their opacity. If during the process a line either appears or disappears, it is an artefact. Since we cannot find any artefacts, we conclude that the approximated matrix is a perfect copy of the original. We suppose that the result obtained evaluating the 2-norm of the error has that value due to the intensity differences we were discussing before.

If we want to obtain a more similar approximation, we can act on the parameters that define the model. For what we have discussed before, we could increase the frequency resolution and the frame resolution to obtain a more accurate activation and template matrixes.

Question Three:

In this point, we will create the mask with which extract the different voices of the Chorale.

First of all we define a function that split the annotation lists in four different lists: one that contains just the annotations related to the soprano line, one the annotations related to the alto, one to the tenor and the last to the bass. This is done just by appending the annotations that share the same label on a same list.

After we defined the function, we call it. Time for check: we print the shape of the soprano annotation list. The result is 54. If we count the number of lines related to the soprano voice in the activation matrix, we note that are actually 54. Great result.

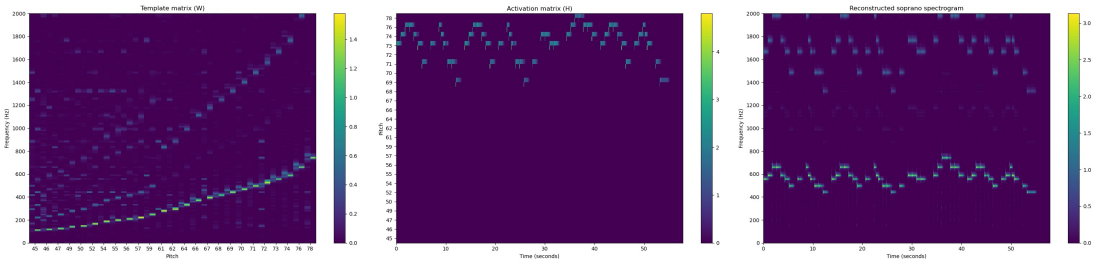
We are ready now to obtain the spectral masks for each sources. So we call the function `initialize_activation` but, this time, with a different annotation set: we use the annotation sets defined in the point before therefore the activation matrix will have as "activated" pitch just the one related to the voice specified by the annotation we are using. Given the fact that each voice has its own pitch dynamic, we have also to specify the `pitch_set` in order to use the same **`template matrix`** we used before. We remain that we have to multiply the two matrixes so first dimension of the activation matrix (that is the pitch set) has to be equal to the zero dimension of the template matrix. We check this condition by printing the shape of activation matrix of the soprano line, that is exactly equal to the one obtained in the previous point. Using the same `pitch_set` is useful also to compare the results.

We create the final template matrixes by evaluating the Hadamard product between the matrixes we have already obtained and the **`activation matrix`** obtained in the previous point. In this way we are

logically masking the **`activation matrix`** since the only elements that survive after the operation are the one that we do not multiply to zero.

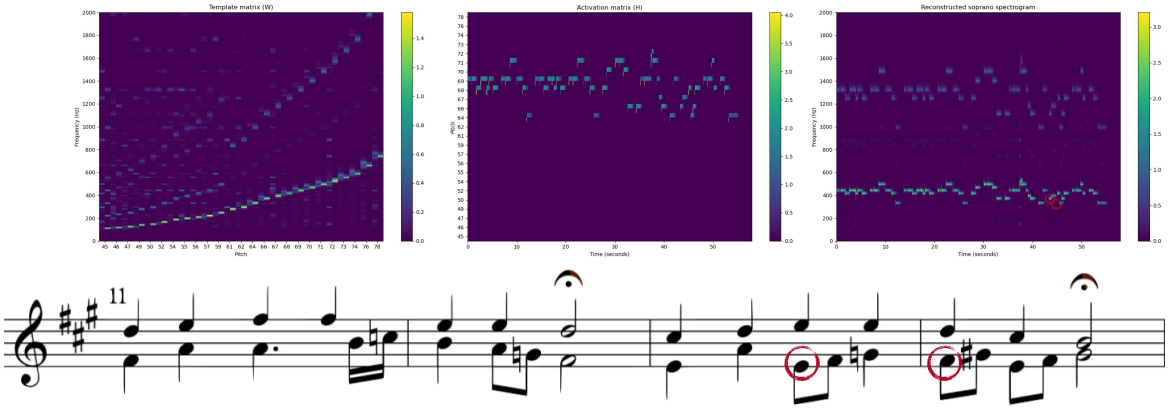
We reconstruct the four spectrograms by algebraically multiplying the activation matrixes to the **`template_matrix`**. Then we plot the results that we are going to analyse.

Soprano part:



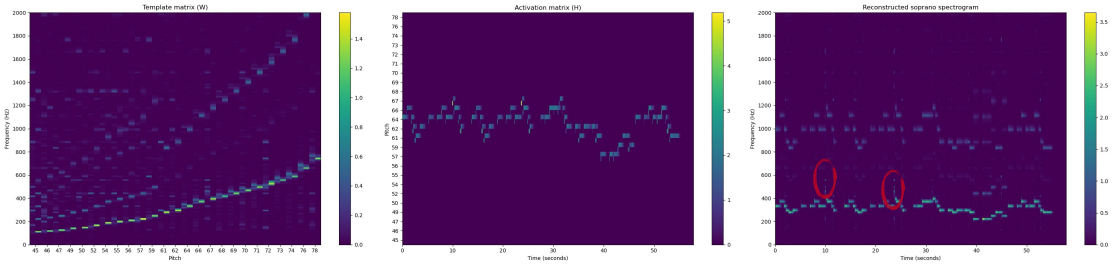
As we can see, in the activation matrix are present just the soprano notes since the other ones have been removed by the masking procedure. In the reconstructed spectrogram we can appreciate the cleanliness of the part: the two sets of frequency are exactly the activation matrix's copy. We expected this result since the pitches used for the soprano voice (from 69 to 78) posses only two harmonics in the template matrix.

Alto part:



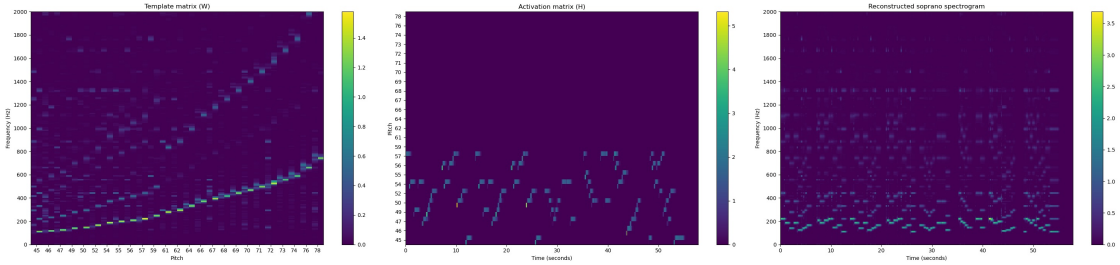
Even in this case, the result looks tidy enough to understand the movement of the part. Just by looking at the sheet music, we can notice that in the final part there are some artefacts and missing notes that we have encircled.

Tenor part:



The tenor part shows even more artefacts especially for those notes whose pitch is below 61. The presence of more harmonics makes the separation more difficult to perform.

Bass part:



The reconstructed spectrogram of the bass part does not look so clear. We suppose that this is why the template matrix is full of harmonics since all the notes of that part are associated to a pitch that goes below 57. As we can see from the template matrix, the notes characterised by those pitches have a very dense spectra. The risk is that with this kind of factorisation we are not able to perfectly extract the main line without introducing artefacts.

Question Four:

As we have just seen, this method introduces a lot of artefacts. To remove them, we apply another kind of masking called *soft masking*. We evaluate the Hadamard division (the element-wise division) between the result just obtained in the point before and the **`V_approx`** matrix to which we add the machine precision. This, because we want to avoid to divide by zero. If we write in formulas the procedure we have:

$$M_{voice} = \left(WH_{voice}\right) \oslash (WH + \epsilon)$$

In this way, every cell of the masked matrix is relativised to the value contained in the same cell of the **`V_approx`** matrix. It is a sort of smoother masking operation.

We apply the method to each voice, we compute the inverse STFT and, after plotting the wave forms, we listen to the results by means of the function `Audio` contained in the library `ipd`.

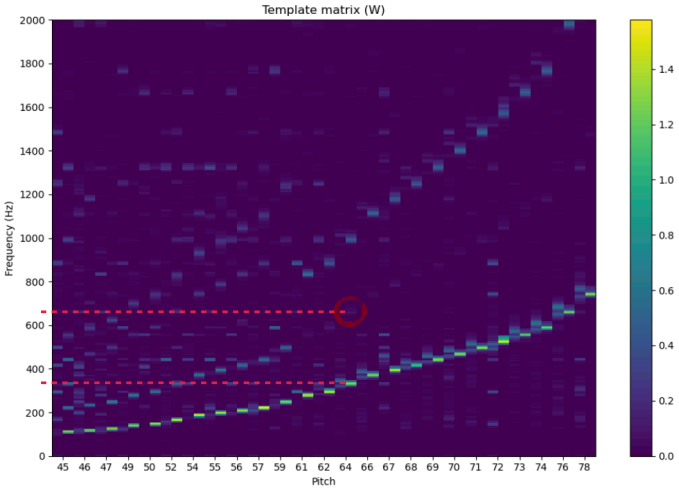
Let us analyse the results:

Soprano part: the soprano part is perfectly extracted from the Chorale. The melody is clear and there are not artefacts if we do not consider the little leak we hear at the beginning of each note. It may be ascribed to a little delay between the actual starting point of the note and the “start” value contained in the annotation. We also observe that there are some missing notes respect to the sheet music, but we cannot blame our algorithm since those notes are not present also in the polyphonic version of the Chorale.

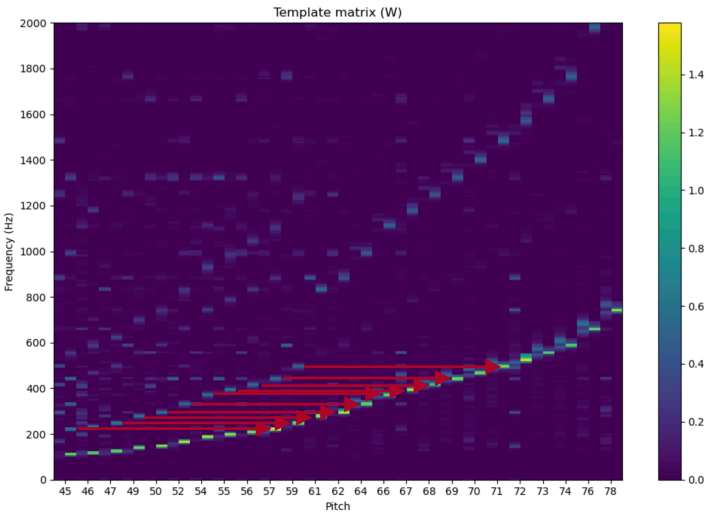
Alto part: the alto part is also well extracted but, if we hear it carefully we can notice a sort of ethereal memento of the soprano part. We suppose that this is because their “pitch recipe” contain also the frequencies related to the soprano voice. Additionally, in this part we can hear a heavy glitch in the fourth movement of the eleventh bar. We suppose that this problem is related to the tolerance we have chosen for the onset given the fact that we bump

into this problem when there are two semiquavers after a dotted note. It seems that algorithm is looking for something that could fill the silence on the downbeat. Even in this case, there are some missing note, but this is not the algorithm's fault.

Tenor part: in the tenor part, the presence of artefacts becomes heavier. The part is still recognisable but we can hear the presence of the other voices each time there is an octave or a fifth between the tenor note and the note of another voice. We can hear this phenomena right at the beginning of the piece, with the E in the third movement of the first bar. We try to justify this effect observing that the second and the third frequency in the Fourier transform has a relation, respectively, of octave and of fifth with the fundamental (if we want to be more precise, the fifth of the octave). So, it is if they would be considered as timbre elements in the **`template matrix`**. In fact, if we look at the **`template matrix`**, we notice that that note (pitch = 64 and fundamental frequency equals to $329.628\ Hz$) presents a line in correspondence of its octave $659.255\ Hz$. Even though it is extremely soft, this is the only way to justify this effect since in the activation matrix there is anything in correspondence of the pitch related to $659.255\ Hz$ (76). Also in this case we can hear artefacts on the downbeat between a dotted note and a couple of semiquavers as we can hear in the fourth movement of the third bar. Also this time, we have some missing notes.



Bass part: the bass part is the worse. Almost every note presents a souvenir of polyphony. This is because, having lower fundamental frequencies, the harmonics in the **`template matrix`** are closer to each other and they fall within the range where the fundamental of the notes of other voices are present. For this reason, those harmonics that are heavily present in the “pitch recipe” of the basso notes make the part sounds still polyphonic. The only exception is the A#2 (pitch 46) since there is no A#3 in the piece. We conclude that, due to the fact that the distance between the harmonics decreases as the frequency decreases, this method works perfectly for high notes and becomes as worse as the pitch goes down.



As last thing, we can watch at the wave forms noticing how the single parts are thinner respect to the original: if we superpose the plot of the four parts we obtain exactly the original wave form.

Question Five:

We pack now all the functions in a single one called `separate` that takes as input the audio file, the annotation data, the length of the fft and the hop size of the fft. In this way, we can perform source separation by applying a soft masking to each signal we put as input.

We define a data metric function that evaluates:

- the 2-norm between the original and the reconstructed signal
- the signal to distortion ratio.

We evaluate this last parameter by means of the function `bss_eval_sources` from of the object `separation` from the library `mir_eval`. This function takes as input an array that contains the reference source, an array that contains the estimated one and a boolean parameter to choose if we want to permute the estimate with the source. As we can reed from the data sheet: “Passing `False` for `compute_permutation` will improve the computation performance of the evaluation; however, it is not always appropriate and is not the way that the BSS_EVAL Matlab toolbox computes bss_eval_sources" so we decide to pass `True`. With other results, it returns the vector of signal to Distortion Ratios (SDR). The formula is the following:

$$SDR \overset{def}{=} 10\log_{10}\left(\frac{\|x\|^2}{\|e_{interferencies} + e_{noises} + e_{artefacts}\|^2}\right)$$

Where $e_{interferencies} + e_{noises} + e_{artefacts} = \hat{x} - x$. So, it tells us how much the reconstructed signal is clean and pure respect to the original; how much it resemble the original. This kind of metric is highly praised by analysts of music information retrieval field because the ratio is normalised only on artefacts, noises and interferences without taking into account the signal itself. This means that it shows the purity of the signal aside from the signal itself and it provides a valid parameter to compare the quality of a certain

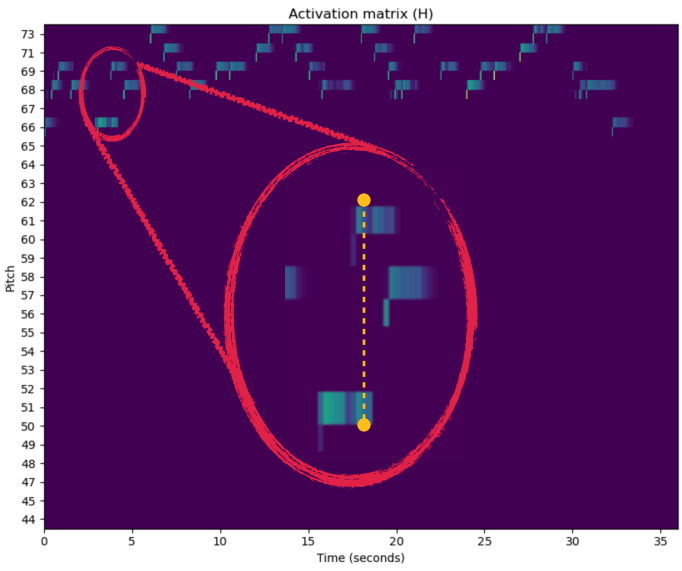
result to other ones. Watching at the statistics, we can consider an optimum result, an SDR around nine considering that the most powerful algorithm written up to now for source separation from a real song reaches an SDR equal to 9.20 dB. Since the organic of the pieces is very simple, we could expect to reach also higher values.

First chorale (*Jesu, meiner Seelen Wonne, J.S. Bach*):

We have deeply analysed the results from the first chorale in the fourth question. We just dwell some seconds on the data metric results to see if they confirms the conclusions we reached. We observe that the 2-norm value increases as we pass from soprano to bass. On the contrary, the SDR decreases. These numbers are perfectly in line with what we have previously discussed: the results of the extractions of the lower voices are dirtier than the ones of the higher voices.

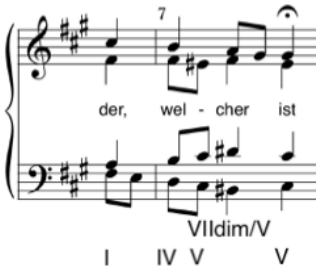
Second chorale (*Seid froh dieweil, J.S. Bach*):

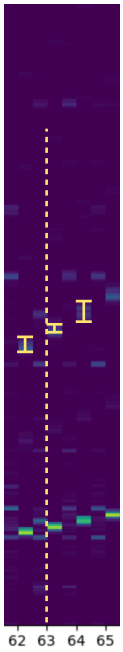
Soprano part: the extraction of the soprano part sounds not too much sully. We can perfectly hear two leaks at the turn of the first and the second bar (without counting the upbeat). We suppose that the problems lies in the fact that we reach a unison at the end of the first semi-phrase. This creates some issues in the activation matrix since it is more difficult to distinguish the two voices. In fact, as we can see, after the unison there is an overlapping of two voices at the next time instant.



Alto part: the alto part presents the same problem of the soprano one especially in the progression that starts from the last movement of bar 8. The note after a unison sounds always a bit dirty. Moreover there are some leaks around the track.

Tenor part: also in this track we can hear some leaks especially during the progression. An interesting thing we can notice is how the D# in the seventh bar is the cleanest note of all the track. This is because this note is out of the harmony of the piece since, in that part, there is a small modulation section called tonicisation. The first chord (F#-) is the tonic one I. From that, we go to the subdominant chord IV^6 (B-). The next chord (C#7) is the dominant V^7 of F#-. In seventh bar's second movement, we bump into a peculiar chord: B#dim7 that is the "diminished chord's dominant" of the home key. The process is called tonicisation since we consider as tonic C#7 for just a few chords. This is why we have that D# in tenor track only in that chord. We can check that by looking at the template matrix: the D# note (pitch = (63)) presents not much harmonics.





Bass part: the bass part sounds like a marching band. Unfortunately, there is the same problem we have identified in the first chorale: the quality of the extraction decreases as we extract deeper voices.

The numbers confirm our analysis but with a curious behaviour. Except from the soprano track, the 2-norm of the error and the SDR show a decreasing trend. This means that the presence of artefacts is getting more and more relevant as we pass from alto to bass but, at the same time, the melody becomes closer to the original.

Third chorale (*Still Nacht, F.X. Gruber*):

Soprano part: given the fact that the rhythmic pattern of this piece is more complex than the one of the previous chorales, the extraction becomes more difficult. We can perfectly distinguish the melody but there is a heavy presence on leaks each time there are semiquavers. As we said before, we can avoid them trying to increase the resolution of the function that evaluate the activation matrix. As we can imagine, the value of 2-Norm is huge and the SDR very low. This means that the track "moves away" from the original melody and the presence of artefacts is still heavy in this first track.

Alto part: the situation is quite similar to the soprano part. Even in this case we have not achieved a good result as it is confirmed by the metrics.

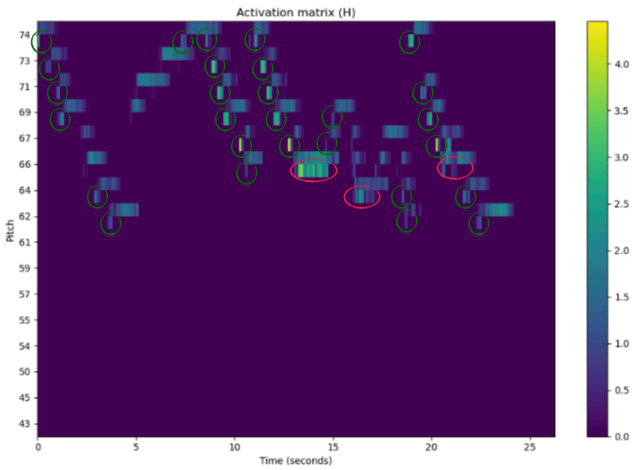
Tenor part: the situation is a bit better for the tenor part since it is made up by just six notes. In spite of that, we can also hear some leaks also during the long sustained part.

Bass part: also the bass part is sully. Furthermore, the presence of lower notes introduce the issue related to the overlapped harmonics: as we have discussed, some of the harmonics that compose the lower pitches overlap the fundamentals of the higher notes. For this reason, it is difficult to obtain a pure sound of them.

This Chorale is not well separated as we can see from the metrics: the lower value of 2-Norm's parameter is never less than 99 Pa.

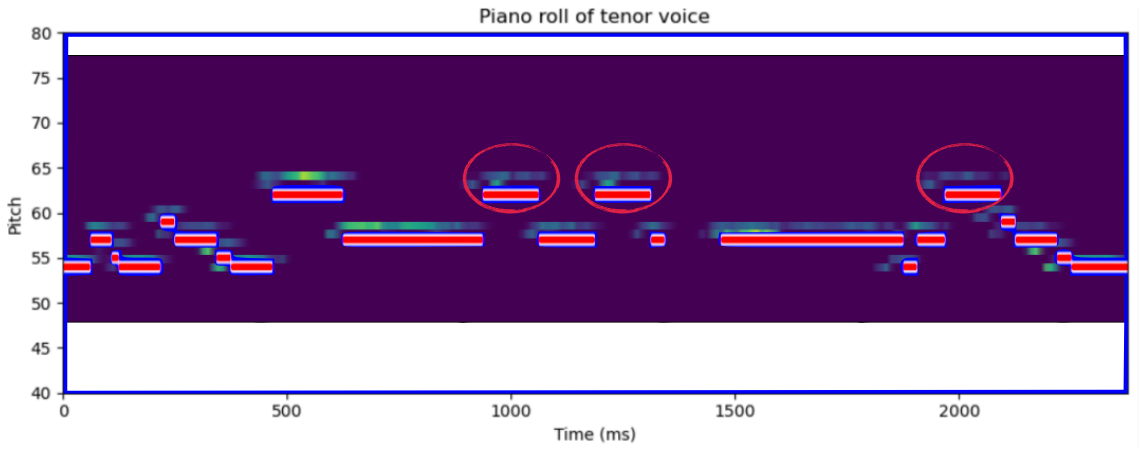
Fourth chorale (*Joy to the world, L. Mason*):

<u>*Soprano part:*</u> in this last chorale, the soprano part is surely clearer than the one in the Stille Nacht, but it is still far from the level of cleanliness reached in Bach's pieces. In the activation matrix are evident both the leaks at the beginning of the note (encircled in green) and the unwanted superpositions (encircled in red).



<u>*Alto part:*</u> also in this case, the part is recognisable but full of leaks.

<u>*Tenor part*</u>: the tenor part would be clearer than the others if it was not for a deleted part from bar 8 to 11. By comparing the piano roll of the tenor midi file with the activation matrix is clear which are the notes we lost. We highlight that the template matrix is shifted-up a bit in order not to make it covered by the piano roll.

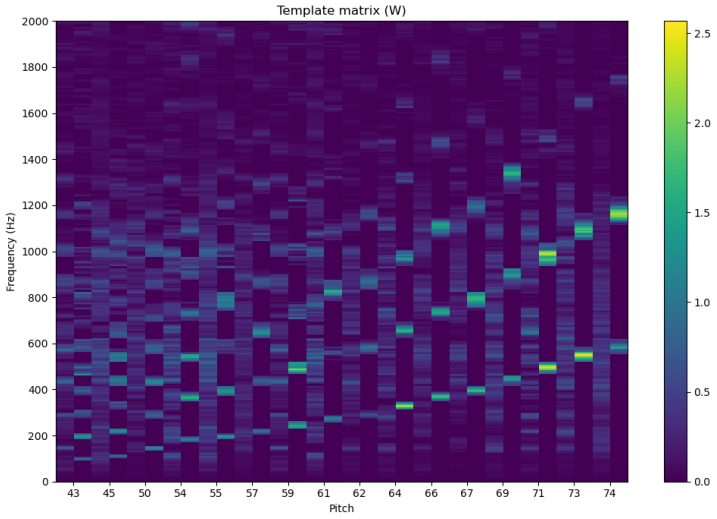


As we can notice, the encircled notes are weaker than the others. Due the artefacts introduced, they are not well distinguishable in the extracted part.

<u>*Bass part*</u>: this time, the bass part sounds as the cleanest. We have to underline that the bass melody is really elementary so it may be just a

sensation to hear it as the cleanest because we don’t have to follow particular movements in the melody. Anyway, also watching the number, it has the lowest 2-Norm value between the other not leading voices.

As last thing regarding this chorale, we have to take a look at the template matrix. As we can see, the frequency distribution for a certain pitch is not clear at all. This could be the cause of the leaks’ heavy presence and the reason why we have so low values of SDR as a consequence.



References:

We reported just the references used for the result's analysis and not the one used to write the code.

1. Colyer Adrian, (2019, 18th February), *The why and how of nonnegative matrix factorisation,* the morning paper, retrieved on 15th December 2023*,* https://blog.acolyer.org/2019/02/18/the-why-and-how-of-nonnegative-matrix-factorization/#:~:text=Nonnegative%matrix%factorization%(NMF)%has,Lee%and%Seung%in%1999
2. Gills N.*, The Why and How of Nonnegative Matrix Factorization,* arXivLabs, 1-2*;* (2014)
3. Y. -X. Wang and Y. -J. Zhang, *Nonnegative Matrix Factorization: A Comprehensive Review*, in IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 6, pp. 1336-1353, June 2013, doi: 10.1109/TKDE.2012.51.
4. Dawn L., (n.d.), *Bach Chorales BWV 359,* retrieved on 15th December 2023, https://www.bach-chorales.com/BWV0359.htm
5. Dawn L., (n.d.), *Bach Chorales BWV 248(3).35(12),* retrieved on 17th December 2023, https://www.bach-chorales.com/BWV0248_35.htm
6. C. Raffel and P.W. Ellis, *Intuitive Analysis, Creation and Manipulation of MIDI Data with pretty_midi*, in 15th International Conference on Music Information Retrieval Late Breaking and Demo Papers, 2014.
7. C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, and D. P. W. Ellis, *mir_eval: A Transparent Implementation of Common MIR Metrics*, Proceedings of the 15th International Conference on Music Information Retrieval, 2014.
8. E. Manilow, P. Seetharaman, J.Salamon, *Basics of source separation - evaluation*, retrieved on 2nd January 2024, https://source-separation.github.io/tutorial/basics/evaluation.html
9. -, *Music Source Separation on MUSDB18*, retrieved on 2nd January 2024 https://paperswithcode.com/sota/music-source-separation-on-musdb18