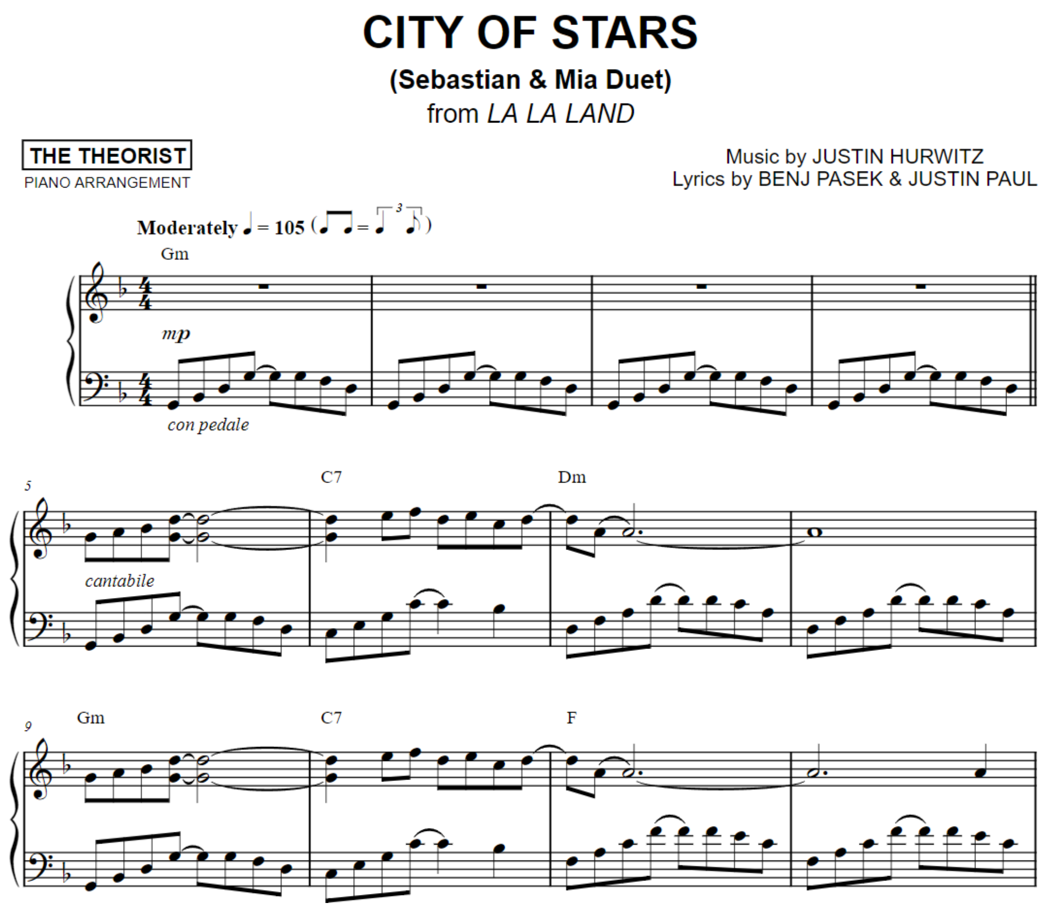
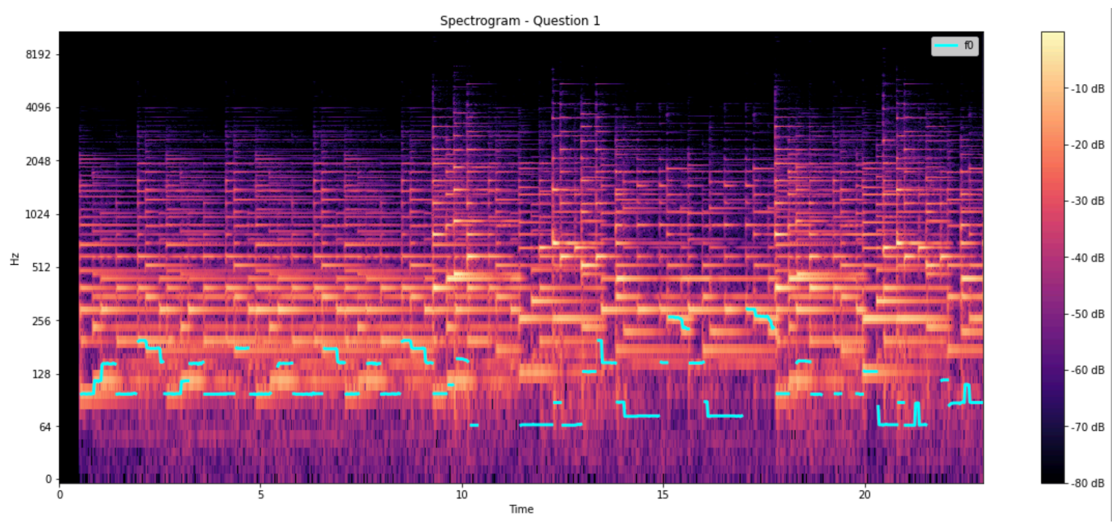
MIR - HW #1

Audio Signal Processing & Musical Key Detection

Question 1.

a.

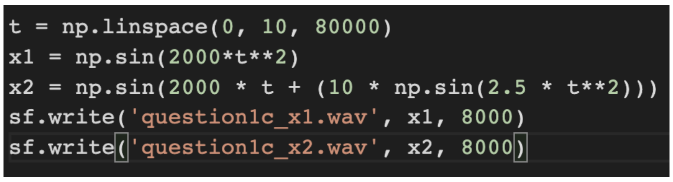




This song[1] is fetched from the theme song of LaLa Land, and it's 36 seconds long. I find a piano cover, so it should be easier to read off the spectrogram. In this recording, we can clearly see the f0 in light blue line and is playing off the left hand part on the music sheet. However, by listening to the song, we can notice that G3 (196.00 Hz) is detected as G2 (98.00 Hz). As right hand joins around 11s, lower frequency is recognized (around C2).
By simply reading the strong note (highlighted in yellow), we can see the melody of both left and right hands. For example, in bar 6 (around 14s), the note is at E5 (659.25 Hz) and F5 (698.46 Hz).

b. Given $x_2(t) = \sin(2000t + 10\sin(2.5t^2))$ and $x(t) = \sin(\phi(t))$, so we know that $\phi(t) = 2000t + 10\sin(2.5t^2)$. Therefore, the IF of x$_2$ will be $f(t) = \frac{1}{2\pi}\frac{d\phi}{dt} = \frac{1}{2\pi}(2000 + 50t\cos(2.5t^2))\ Hz$.

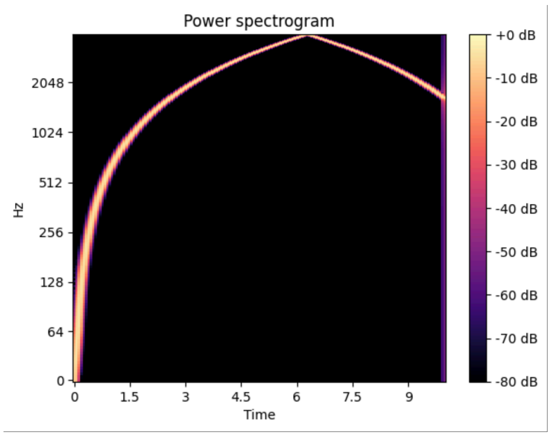
c. Here is how I produce the audios

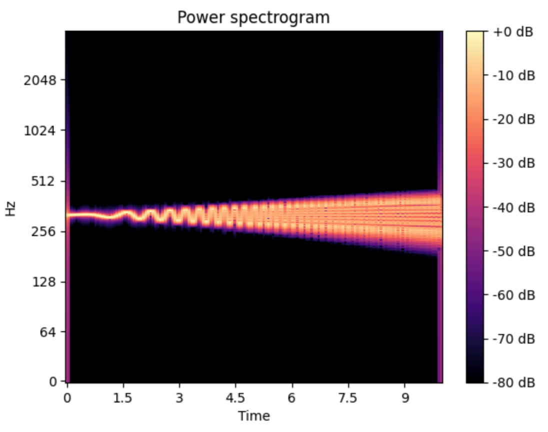


```
t = np.linspace(0, 10, 80000)
x1 = np.sin(2000*t**2)
x2 = np.sin(2000 * t + (10 * np.sin(2.5 * t**2)))
sf.write('question1c_x1.wav', x1, 8000)
sf.write('question1c_x2.wav', x2, 8000)
```

For x1, the sound is increasing throughout the first 6.5 seconds, and decreasing afterward. As for x2, the sound is like oscillating, increasing and decreasing, and is more observable in the end.

d. Here are the two spectrograms for x1 and x2. The first spectrogram does look like how the audio sound to me, but it didn't match the idea of keep increasing like a bird's chirp. The second spectrogram look like how it sound to me. The frequency oscillates throughout that 10 seconds.

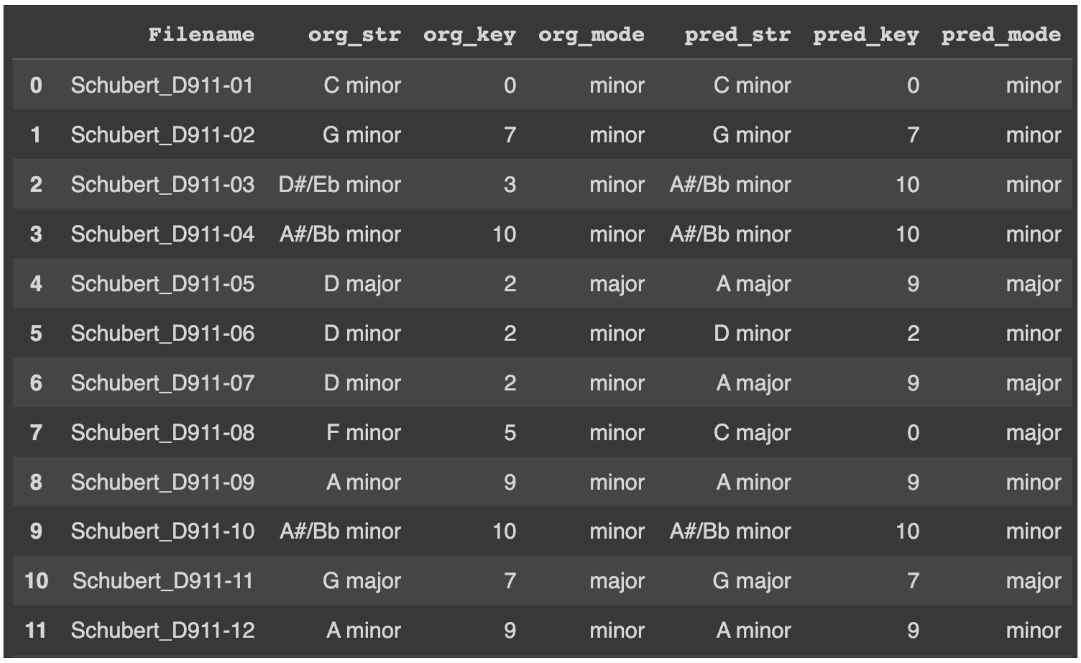




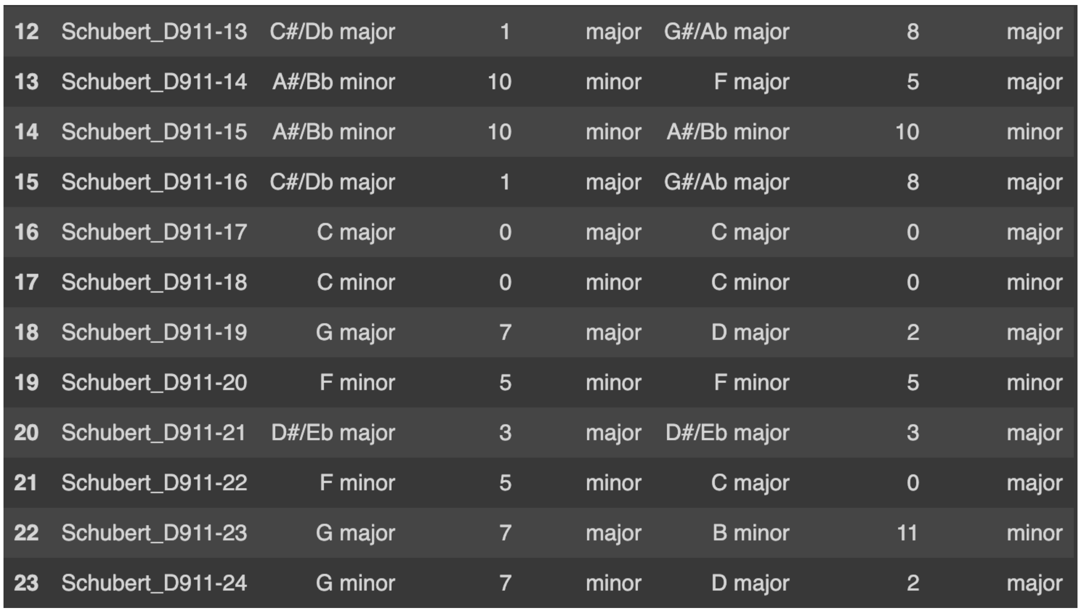
[1] https://www.youtube.com/watch?v=oAXXSFKfjLs&ab_channel=TheTheorist

Question 2.

a. For Global Key detection, I used K-S detection with late fusion. And here is the result I got after the prediction.



	Filename	org_str	org_key	org_mode	pred_str	pred_key	pred_mode
0	Schubert_D911-01	C minor	0	minor	C minor	0	minor
1	Schubert_D911-02	G minor	7	minor	G minor	7	minor
2	Schubert_D911-03	D#/Eb minor	3	minor	A#/Bb minor	10	minor
3	Schubert_D911-04	A#/Bb minor	10	minor	A#/Bb minor	10	minor
4	Schubert_D911-05	D major	2	major	A major	9	major
5	Schubert_D911-06	D minor	2	minor	D minor	2	minor
6	Schubert_D911-07	D minor	2	minor	A major	9	major
7	Schubert_D911-08	F minor	5	minor	C major	0	major
8	Schubert_D911-09	A minor	9	minor	A minor	9	minor
9	Schubert_D911-10	A#/Bb minor	10	minor	A#/Bb minor	10	minor
10	Schubert_D911-11	G major	7	major	G major	7	major
11	Schubert_D911-12	A minor	9	minor	A minor	9	minor

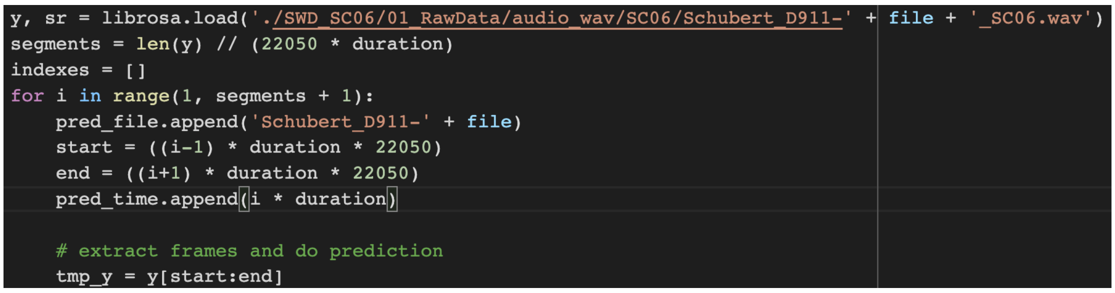


12	Schubert_D911-13	C#/Db major	1	major	G#/Ab major	8	major
13	Schubert_D911-14	A#/Bb minor	10	minor	F major	5	major
14	Schubert_D911-15	A#/Bb minor	10	minor	A#/Bb minor	10	minor
15	Schubert_D911-16	C#/Db major	1	major	G#/Ab major	8	major
16	Schubert_D911-17	C major	0	major	C major	0	major
17	Schubert_D911-18	C minor	0	minor	C minor	0	minor
18	Schubert_D911-19	G major	7	major	D major	2	major
19	Schubert_D911-20	F minor	5	minor	F minor	5	minor
20	Schubert_D911-21	D#/Eb major	3	major	D#/Eb major	3	major
21	Schubert_D911-22	F minor	5	minor	C major	0	major
22	Schubert_D911-23	G major	7	major	B minor	11	minor
23	Schubert_D911-24	G minor	7	minor	D major	2	major

	stft	cqt	cens
RA scores	0.625	0.5833	0.5417
WA scores	0.6875	0.6958	0.6458

It looks like stft have the best result in general, it has the average of 0.656. The CQT and cens method are interesting, it is likely to detect a chord as something else (relative/perfect-fifth/parallel), and this is why they have a large difference between RA scores and WA scores.

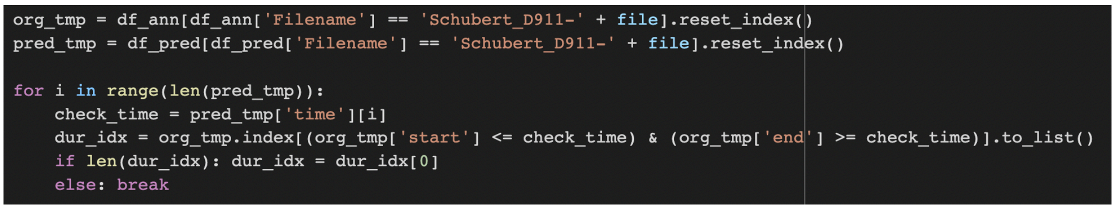
b. For local key detection, I also used the same approach. But for the frames extraction, I use the method as described in the questions with 15 seconds before the detected time and 15 seconds after it, which is 30 seconds in total.



```
y, sr = librosa.load('./SWD_SC06/01_RawData/audio_wav/SC06/Schubert_D911-' + file + '_SC06.wav')
segments = len(y) // (22050 * duration)
indexes = []
for i in range(1, segments + 1):
    pred_file.append('Schubert_D911-' + file)
    start = ((i-1) * duration * 22050)
    end = ((i+1) * duration * 22050)
    pred_time.append(i * duration)

    # extract frames and do prediction
    tmp_y = y[start:end]
```

I found that length of y is multiple of sampling rate and the music length. So I cut the y to a smaller duration (30 seconds) tmp_y.

And for the RA and WA score, I decided to check my estimations one by one with the ground truth file. For example, if the estimation at 10th second is within that (start ~ end) range in the csv file, then I check the estimation correspondingly.



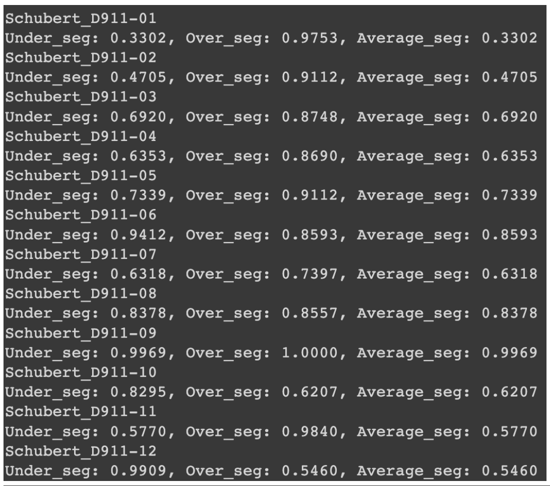
```
org_tmp = df_ann[df_ann['Filename'] == 'Schubert_D911-' + file].reset_index()
pred_tmp = df_pred[df_pred['Filename'] == 'Schubert_D911-' + file].reset_index()

for i in range(len(pred_tmp)):
    check_time = pred_tmp['time'][i]
    dur_idx = org_tmp.index[(org_tmp['start'] <= check_time) & (org_tmp['end'] >= check_time)].to_list()
    if len(dur_idx): dur_idx = dur_idx[0]
    else: break
```

By doing so, I can get the index of the correct frame on the csv file.

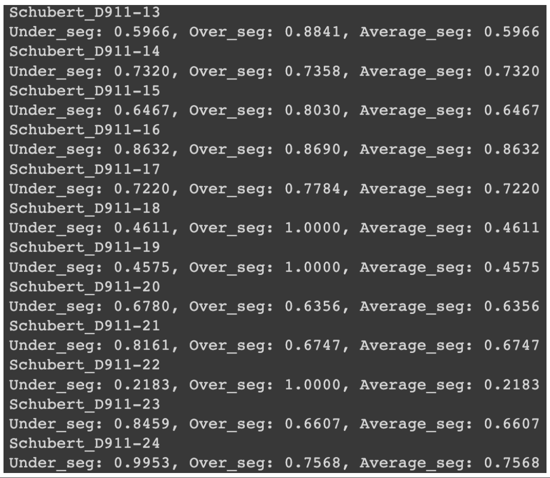
	stft	cqt	cens
RA scores	0.4263	0.4940	0.4741
WA scores	0.5191	0.6080	0.5880

It is interesting that stft performs the best in global key detection while cqt and cens perform better in local key detection. I think they are predicting more accurately on the music with shorter duration.

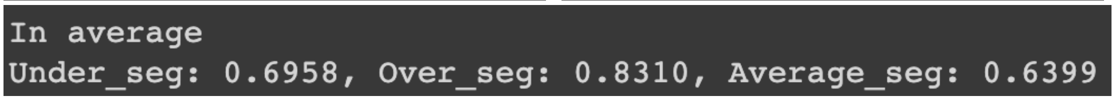
c. I assume that the small segments with the same chord should be in a same segment, so I combine them into a big one and extract the start and end time.



```
Schubert_D911-01
Under_seg: 0.3302, Over_seg: 0.9753, Average_seg: 0.3302
Schubert_D911-02
Under_seg: 0.4705, Over_seg: 0.9112, Average_seg: 0.4705
Schubert_D911-03
Under_seg: 0.6920, Over_seg: 0.8748, Average_seg: 0.6920
Schubert_D911-04
Under_seg: 0.6353, Over_seg: 0.8690, Average_seg: 0.6353
Schubert_D911-05
Under_seg: 0.7339, Over_seg: 0.9112, Average_seg: 0.7339
Schubert_D911-06
Under_seg: 0.9412, Over_seg: 0.8593, Average_seg: 0.8593
Schubert_D911-07
Under_seg: 0.6318, Over_seg: 0.7397, Average_seg: 0.6318
Schubert_D911-08
Under_seg: 0.8378, Over_seg: 0.8557, Average_seg: 0.8378
Schubert_D911-09
Under_seg: 0.9969, Over_seg: 1.0000, Average_seg: 0.9969
Schubert_D911-10
Under_seg: 0.8295, Over_seg: 0.6207, Average_seg: 0.6207
Schubert_D911-11
Under_seg: 0.5770, Over_seg: 0.9840, Average_seg: 0.5770
Schubert_D911-12
Under_seg: 0.9909, Over_seg: 0.5460, Average_seg: 0.5460
```



```
Schubert_D911-13
Under_seg: 0.5966, Over_seg: 0.8841, Average_seg: 0.5966
Schubert_D911-14
Under_seg: 0.7320, Over_seg: 0.7358, Average_seg: 0.7320
Schubert_D911-15
Under_seg: 0.6467, Over_seg: 0.8030, Average_seg: 0.6467
Schubert_D911-16
Under_seg: 0.8632, Over_seg: 0.8690, Average_seg: 0.8632
Schubert_D911-17
Under_seg: 0.7220, Over_seg: 0.7784, Average_seg: 0.7220
Schubert_D911-18
Under_seg: 0.4611, Over_seg: 1.0000, Average_seg: 0.4611
Schubert_D911-19
Under_seg: 0.4575, Over_seg: 1.0000, Average_seg: 0.4575
Schubert_D911-20
Under_seg: 0.6780, Over_seg: 0.6356, Average_seg: 0.6356
Schubert_D911-21
Under_seg: 0.8161, Over_seg: 0.6747, Average_seg: 0.6747
Schubert_D911-22
Under_seg: 0.2183, Over_seg: 1.0000, Average_seg: 0.2183
Schubert_D911-23
Under_seg: 0.8459, Over_seg: 0.6607, Average_seg: 0.6607
Schubert_D911-24
Under_seg: 0.9953, Over_seg: 0.7568, Average_seg: 0.7568
```



```
In average
Under_seg: 0.6958, Over_seg: 0.8310, Average_seg: 0.6399
```

It looks like the segmentations I made were not precise, however, over-segmentation still has a satisfying performance overall.