AMATH 582 Homework 2

Yiyun Dong

Department of Physics
University of Washington
Email: yiyund@uw.edu
Github: yeewantung

Abstract

This project consists of two parts. In the first part, by performing the Gabor transform on a piece of Handel's music, we study the change in the frequencies of Handel's music over time, and observe the effect of different Gabor transform kernels and parameters on the resolution and interpretability of the time-frequency spectrogram. In the second part, we perform the Gabor transform on a music piece called `Mary had a little lamp` played by two different instruments. We use the obtained time-frequency spectrogram to determine the musical notes that appear in two pieces of audio, and then determine the music score of these two pieces of audio.

1 Introduction and Overview

In signal processing, the time–frequency analysis contains the techniques that study a signal in both the time and frequency domains simultaneously [17]. It is widely applied in sampling, optics, acoustics and biomedicine. However, the time-frequency analysis cannot be realized by the Fourier transform, since the Fourier transform can only show the spectrum of a time series but not the frequency of the time series changing over time [11]. The short-time Fourier transform methods are introduced to address this problem [15]. Those methods involve a Fourier transform on a short time section of data to reveal the frequency of occurrence during this time period.

The Gabor transform, named after Dennis Gabor, is a special case of the short-time Fourier transform [12]. It is used to determine the sinusoidal frequency and phase content of local time sections of a signal as it changes over time.

In this project, we implement the time-frequency analysis on two parts. In the first part, we perform the Gabor Transform on a piece of Handel's music. Our focus is which window function and window width can best reveal the relationship between frequency and time in this piece of music. In the second part, we perform the Gabor transform on `Mary had a little lamp` played by two different instruments (piano and recorder). We use the obtained spectrogram to determine the frequency of nodes appeared in the performance and hence reveal the original music scores for two pieces of performance.

2 Theoretical Background

2.1 Gabor transform

As we learned from our textbook [1], the Fourier transform decomposes a time-domain signal into a frequency distribution in the frequency domain.

$$\begin{aligned} F(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x)\, dx \\ f(x) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k)\, dk \end{aligned} \tag{1}$$

In practice, the Fourier transform of data can be implemented in computer by an algorithm called the Fast Fourier transform, with time complexity as $\mathcal{O}(N \log N)$ [10].

The Fourier transform is one of the most important methods for analysis of signal. However, the Fourier transform has limitations on time-frequency analysis. At the meantime of revealing all the frequency information, the Fourier transform loses all the time information and therefore we cannot tell at a sound wave of certain frequency occurs at what time [1]. The reason of this limitation is that the frequency and time of an object cannot be accurately measured simultaneously, known as Heisenberg's uncertainty principle [13].

In order to be able to know which frequencies exist in a certain period of time, we need to localize the data, and then focus on the Fourier transform of this period of time. Gabor introduces a method that localize both time and frequency information [1]. This method uses a sliding window to keep a short time period of data then applies Fourier transform. This method is called the Gabor transform, also known as the Fourier transform.

$$\begin{aligned} g_{t,\omega}(\tau) &= e^{i\omega\tau} g(\tau - t) \\ \mathcal{G}_{[f]}(t,\omega) &= \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau}\, d\tau = (f, \bar{g}_{t,\omega}) \end{aligned} \tag{2}$$

Notice that the window function is centered at time t. By sliding the window, *i.e.* moving the center t, the data of different time period are obtained and extracted the frequency information. Thus we can obtain the frequency distribution associated with different t instant and realize what frequency occurs at a certain time [1].

During localizing in time, it is determined by the Heisenberg's uncertainty principle that some frequency information has to be sacrificed to keep time information from completely losing [13]. The wavelength of the Fourier modes cannot be larger than the size of the window. Thus, information on small frequencies are lost in the Gabor Transform. The resolution of the frequency will be low if we exceedingly localized time. On the other hand, if the window width is too large, most time information will be lost and we will not be able to tell what frequency occurs at a certain time. Therefore, we have to investigate different window function and window width, in order to determine which configuration optimizes the performance of time-frequency analysis.

2.2 Overtones and timbre

The instruments for classical music, just as most sounds in nature, are produced by mixing many standing waves with different frequencies [8]. Those standing waves with different frequencies are called *complex tones.* The sound produced by a single frequency is called *pure tone.* Knocking on a tuning fork or playing with an electronic synthesizer, we can hear this kind of sound that does not exist in nature. A pure tone is relatively boring. Among the multiple sound waves emitted by an instrument, the most easily heard one is the sound from the standing wave with the lowest frequency. It determines the pitch of this sound, which is called *fundamental* [8]. In addition, the sounds from other standing waves with higher frequencies (pitches) are called *overtones* [8]. Overtones have the integer multiples of the frequency of the fundamental and usually have smaller amplitude than the fundamental [14].

As we learned from the web, the *timbre* of an instrument is determined by its overtones [16]. If we listen to each of the fundamental and overtones separately, we will hear a boring sound, that is similar to the sound of a buzzer. But if we listen the fundamental and overtones together, the sound will become rich and pleasant. This exactly explains how the overtones affect the *timbre* of an instrument. Different instruments have different vocal materials and resonance methods. When the same pitch is played, the energy distribution of the fundamental and overtones is different in different instruments. When the same instrument plays different pitches, the energy distribution of the fundamental and overtones is similar, and thus the tone of the instrument is maintained the same.

We'll further discuss overtones in the second part of computational result.

3 Algorithm Implementation and Development

1. At each time slide of time span, use a window to keep the center information of audio data. Window size is adjustable.
2. Apply short time Fourier Transform to the windowed data. This is called Gabor Transform.
3. Apply a filter function if needed to filter overtones in the frequencies of audio data.
4. Save the frequency distribution.
5. Plot the windowing process if needed.
6. After completing the above steps for each time slide, plot the frequency distribution versus time. This is a two-dimensional (2D) spectrogram.

Algorithm 1: Gabor Transform for Time-Frequency Analysis

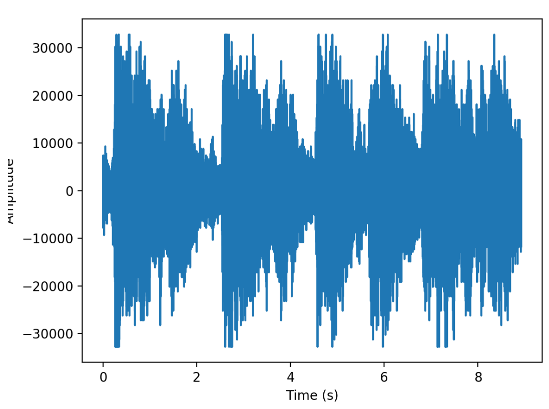
```
Import data from an audio file
for $t_j$ in time span of audio data do
    Apply window function $g$ centered at $t_j$ to data
    Apply Fourier Transform to windowed data
    if Filter overtones then
        Apply filter function filter_func to the Fourier Transformed data
    end if
    Save the frequency distribution in spc[j]
    if Plot then
        Plot data - time and window function $g$
        Plot windowed data - time
        Plot Fourier Transformed data - frequency
    end if
end for
Plot the Time-Frequency spectrogram spc
```

4 Computational Results

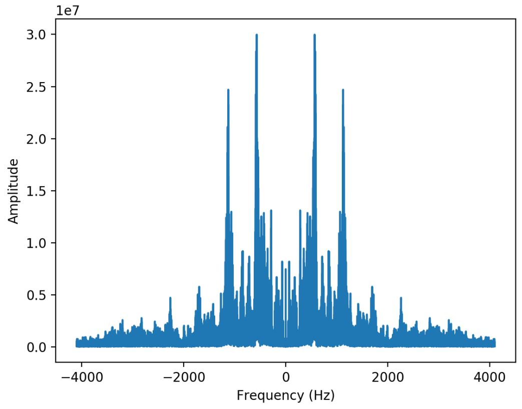
4.1 Part I

In this part, we investigate a 9-seconds piece of Handel's music. The time series and the Fourier transform of this audio piece are shown in Figure 1.

In order to capture the distribution of frequency at each time instant, the Gabor transform is applied to this audio piece, as shown in Figure 3. However, as discussed in section 2, there is a trade-off between the accuracy of time information and frequency information. The resolution of the frequency will be low if we exceedingly localized time. On the other hand, if the window width is too large, most time information will be lost and we will not be able to tell what frequency occurs at a certain time. Here we use three different window functions (Gaussian window, Mexican hat window and Shannon window) and three different window widths, in order to determine which configuration optimizes the performance of time-frequency analysis.



(a) Amplitude of time seires



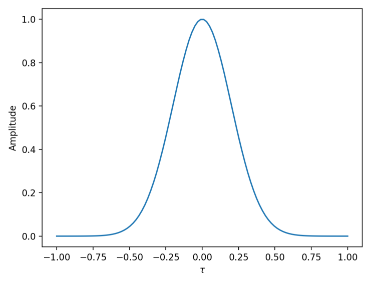
(b) Frequency distribution

Figure 1: The time series and the Fourier transform of Handel's music

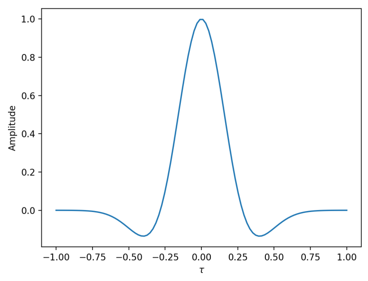
The function forms of Gaussian window, Mexican hat window and Shannon window are

$$\begin{aligned} & g_{t,\omega}(\tau) = e^{i\omega\tau} g(\tau - t) \\ \text{Gaussian:} \quad & g(\tau - t) = \exp(-\frac{(\tau-t)^2}{2\text{width}^2}) \\ \text{Mexican hat:} \quad & g(\tau - t) = (1 - \frac{(\tau-t)^2}{2\text{width}^2})\exp(-\frac{(\tau-t)^2}{2\text{width}^2}) \\ \text{Shannon:} \quad & g(\tau - t) = \begin{cases} 1 & ,|\tau - t| \leq \text{width} \\ 0 & ,|\tau - t| > \text{width} \end{cases} \end{aligned} \tag{3}$$

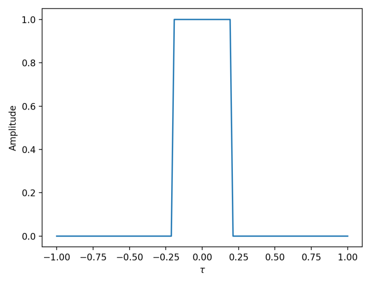
Figure 2 shows the shape of Gaussian window, Mexican hat window and Shannon window.



(a) Amplitude of time seires



(b) Frequency distribution

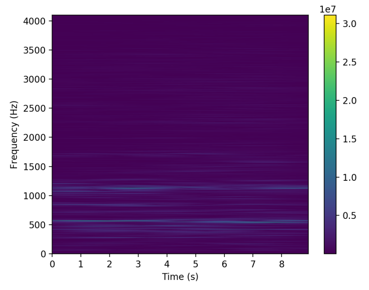


(c) Frequency distribution

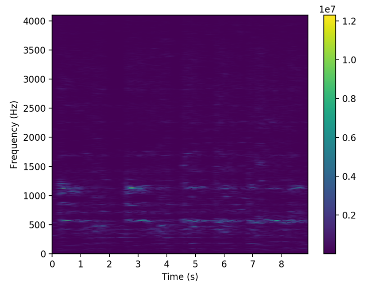
Figure 2: The time series and the Fourier transform of Handel's music

There window widths used here are $0.01, 0.2$ and 1. The resulting spectrograms are shown in Figure 3. As predicted, the frequency resolution is high and the time resolution is low when using wide window. It is not clear to us how the amplitude of the audio changes with time, whereas the frequency information is localized. This is the case of undersampling. On the other hand, the time resolution is high and the frequency resolution is low when using small window. The frequency peaks go blurrier and more spread out, and hence we cannot tell which exact frequency happens at each time instant. This is the case of oversampling. With the medium wide window and a good choice of window function imposed on the data, we can obtain spectrograms with satisfying time and frequency resolution.

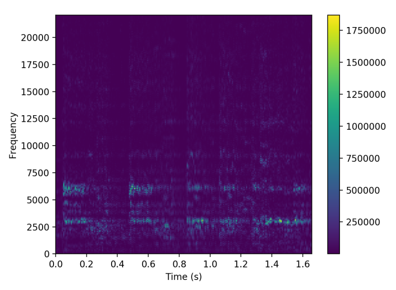
Notice that the spectrogram using the Shannon window presents a checkered distribution both in time and frequency, which cannot reflect the continuity of intensity. It can also be seen that some unreal or discontinuous frequencies appear in the spectrogram using the Shannon window. Spectograms using the Mexican hat window or the Gaussian window have higher resolution than Shannon window, and the intensity is continuous in both frequency and time. I would say that the spectrogram with the Mexican hat function and medium width of window possess the optimal time and frequency resolution, since the time resolution of the spectrogram using the Mexican hat window seems to be higher than the one using the Gaussian window.



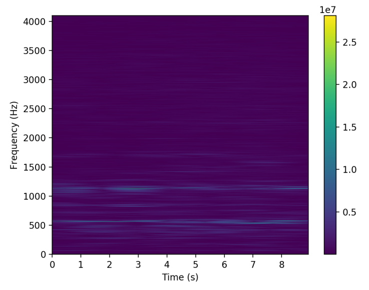
(a) Gaussian wide window



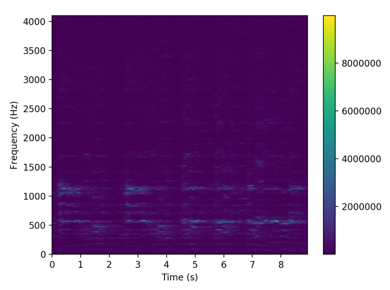
(b) Guassian medium wide window



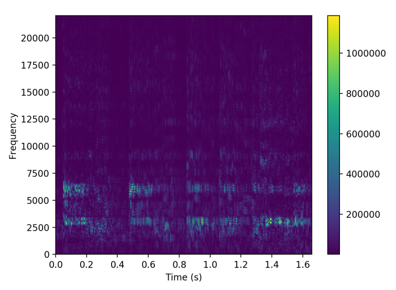
(c) Gaussian small window



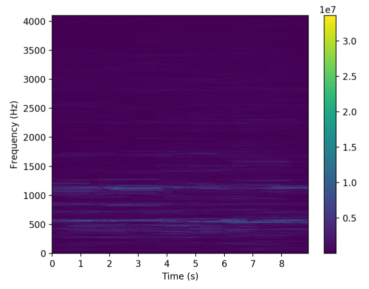
(d) Mexican wide window



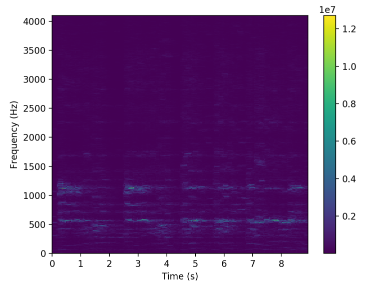
(e) Mexican medium wide window



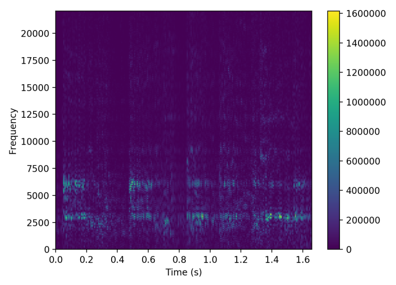
(f) Mexican small window



(g) Shannon wide window



(h) Shannon medium wide window



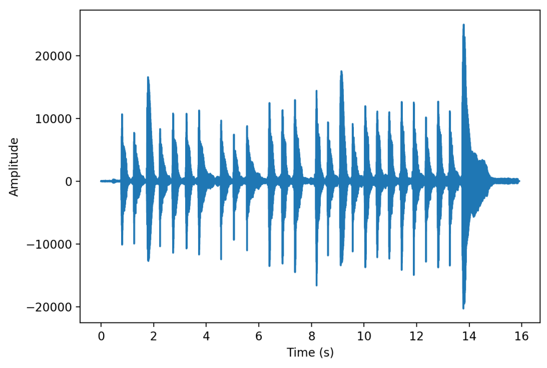
(i) Shannon small window

Figure 3: Time-frequency spectrogram of Handel's music achieved by the Gabor transform with different window functions and window widths

4.2 Part II

In this part, we investigate two music pieces of `Mary had a little lamp` played by two instruments (piano and recorder). The time series and Fourier transform of the time series are respectively shown in Figure 4,5.

Note that there are three main peaks appeared in each Fourier transform (Figure 5). In addition, there are several secondary peaks with frequencies corresponding to the integer multiples of the three main peaks. Those secondary peaks are overtones, as we mentioned in section 2. The reason why the timbre of the piano and recorder are different lies in their overtones. We look the spectrum of two instruments playing the first note of Mary had a little lamp, as shown in Figure 6. The fundamentals of two instruments are marked as "1", whereas the overtones are denoted by other numbers. Notice that the peak "2" and peak "3" of the


(a) Piano

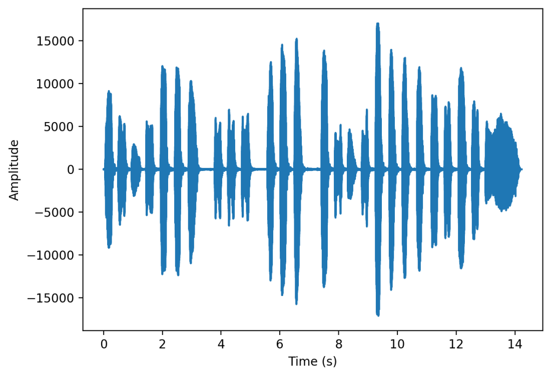
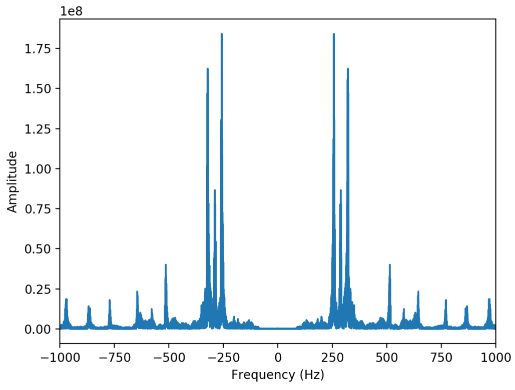

(b) Recorder

Figure 4: Amplitude versus time of `Mary had a little lamp` played by two instruments


(a) Piano

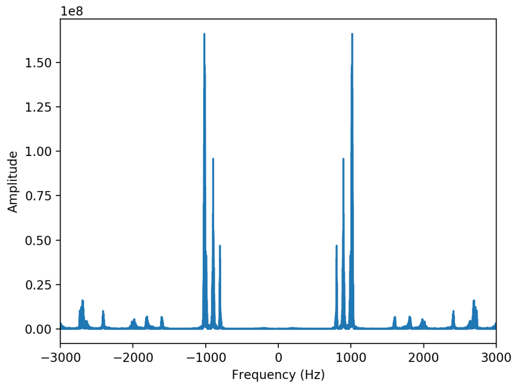
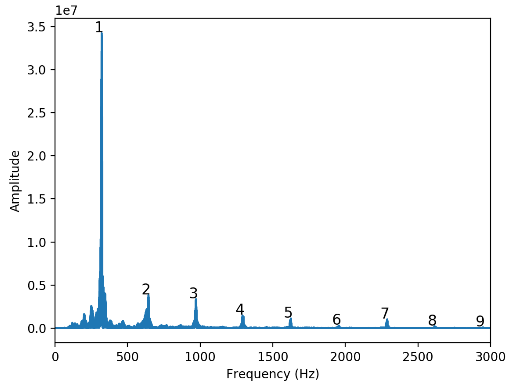

(b) Recorder

Figure 5: Fourier transform of `Mary had a little lamp` played by two instruments


(a) Piano

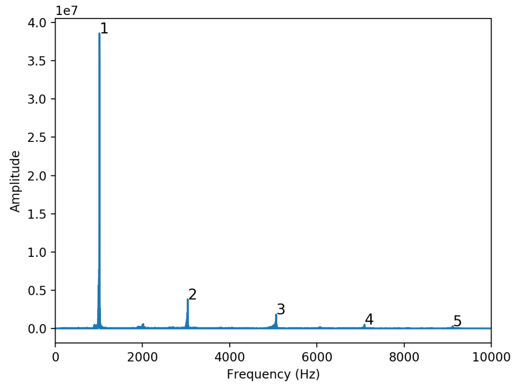
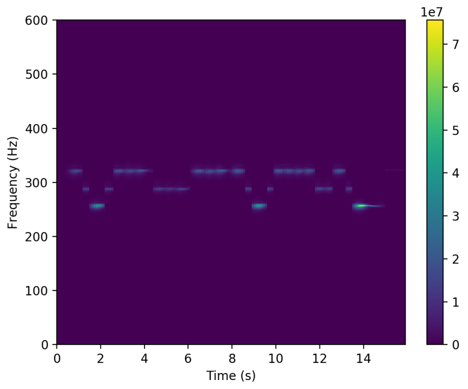
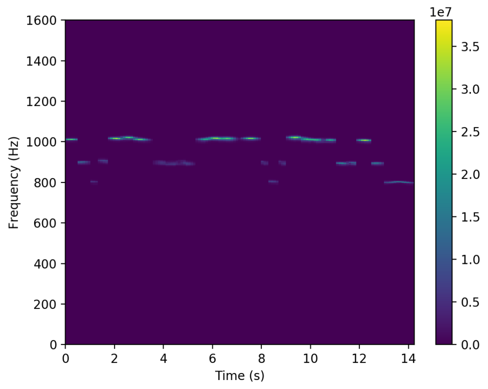

(b) Recorder

Figure 6: Fundamental and overtones of `Mary had a little lamp` played by two instruments

spectrum of piano have similar intensity, whereas the peak "3" of the spectrum of recorder is obviously lower than the peak "2". Besides, except for the first three, the other overtones of recorder are very weak. As a comparison, we can see that the overtones of piano up to 9 times the fundamental frequency still have a considerable intensity. This is the reason why we can distinguish the piano and recorder by their *timbre*.



(a) Piano

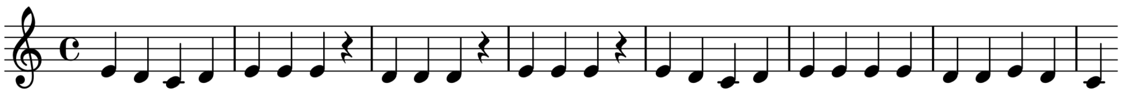


(b) Recorder

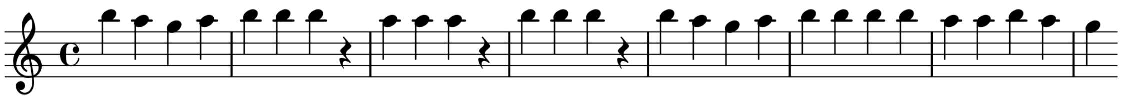
Figure 7: Time-frequency pectrogram of `Mary had a little lamp` played by two instruments

We then apply the Gabor transform to two musical pieces and obtain the spectrograms, as shown in Figure 7. To eliminate the overtones, we apply a gaussian filter centered at the main peak to each Fourier tranform of windowed data, so that only the strongest peak left at each time instant in the spectrograms.

Both the spectrograms of the performance by piano and recorder show three main frequencies. The frequency for three nodes played by the piano are about $316\,Hz$, $282\,Hz$, $251\,Hz$, respectively corresponding to $E5$ ($329.63\,Hz$), $D5$ ($293.66\,Hz$) and $C5$ ($261.63\,Hz$). The frequency for three nodes played by the recorder are about $1000\,Hz$, $880\,Hz$, $789\,Hz$, respectively corresponding to $B5$ ($987.77\,Hz$), $A5$ ($880.00\,Hz$) and $G5$ ($783.99\,Hz$). The spectrograms also show the pitch of sound at different times. The occurrences of three nodes in two cases are then used to create two music score sheets, as shown in Figure 8.



(a) Piano



(b) Recorder

Figure 8: Music score sheets of `Mary had a little lamp` played by two instruments

5 Summary and Conclusions

In this project, we use the Gabor transform to process three audio pieces. In the first part, Handel's music, we study the effect of choosing different window functions and window widths on time-frequency analysis. The conclusion is that a moderate window width and Mexican hat window or Gaussian window as window function can give us a spectrogram with satisfying time resolution and frequency resolution.

In addition, using small translations of the Gabor window leads to oversampling, while using coarse/large translations of the Gabor window lead to undersampling. In general, I am more inclined to accept undersampling when it cannot be accurately determined what window width to choose. This is because our original intention is to perceive the frequency distribution over time. If it is impossible to distinguish which frequencies exist, such as in the case of oversampling, then our time-frequency analysis is meaningless.

In the second part, we use the Gabor transform to acquire the music scores corresponding to the audio piece of `Mary had a little lamp` played by two different instruments. We confirm that the Gabor transform is useful in time-frequency analysis. When the Gabor transform is applied to extract the frequency information of a musical signal, we are able to know the musical notes played and the occurrences and distribution of overtones in the musical signal.

In conclusion, the Gabor transform is powerful in time-frequency analysis, and it is important to choose an appropriate window function and a moderate window width during the Gabor transform.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

[2] *Lilypond Reference.* URL: `http://lilypond.org/`.

[3] *Matplotlib Pcolormesh Reference.* URL: `https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.pcolormesh.html`.

[4] *Matplotlib Reference.* URL: `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html`.

[5] *Mingus Reference.* URL: `https://bspaans.github.io/python-mingus/index.html`.

[6] *NumPy Reference.* URL: `https://docs.scipy.org/doc/numpy/reference`.

[7] *Simpleaudio Reference.* URL: `https://simpleaudio.readthedocs.io/en/latest/simpleaudio.html`.

[8] *Timbre.* URL: `https://www.jianshu.com/p/c5ce10f1a8e9`.

[9] *Wavfile I/O Reference.* URL: `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.io.wavfile.read.html`.

[10] *Wikipedia - Fast Fourier transform.* URL: `https://en.wikipedia.org/wiki/Fast_Fourier_transform`.

[11] *Wikipedia - Fourier transform.* URL: `https://en.wikipedia.org/wiki/Fourier_transform`.

[12] *Wikipedia - Gabor transform.* URL: `https://en.wikipedia.org/wiki/Gabor_transform`.

[13] *Wikipedia - Heisenberg uncertainty principle.* URL: `https://en.wikipedia.org/wiki/Uncertainty_principle`.

[14] *Wikipedia - Overtone.* URL: `https://en.wikipedia.org/wiki/Overtone`.

[15] *Wikipedia - Short-time Fourier transform.* URL: `https://en.wikipedia.org/wiki/Short-time_Fourier_transform`.

[16] *Wikipedia - Timbre.* URL: `https://en.wikipedia.org/wiki/Timbre`.

[17] *Wikipedia - Time-frequency analysis.* URL: `https://en.wikipedia.org/wiki/Time%E2%80%93frequency_analysis`.

Appendix A Python Functions used

Below are the Python functions implemented in the code with a brief explanation.

- `numpy.fft.fft(a)` compute the one-dimensional discrete Fourier Transform of `a` [6].
- `numpy.fft.ifft(a)` compute the one-dimensional inverse discrete Fourier Transform of `a` [6].
- `numpy.fft.fftshift(x, axes=None)` shift the zero-frequency component to the center of the spectrum of `a` [6].
- `matplotlib.pyplot` is a state-based interface to matplotlib [4].
- `simpleaudio.play_buffer(audio_data, num_channels, bytes_per_sample, sample_rate)` starts playback of audio data from an object supporting the buffer interface and with the given playback parameters [7].
- `Axes.pcolormesh(self, *args, alpha=None, norm=None, cmap=None, vmin=None, vmax=None, shading='flat', antialiased=False, data=None, **kwargs)` creates a pseudocolor plot with a non-regular rectangular grid [3].
- `scipy.io.wavfile.read(filename, mmap=False)` returns the sample rate (in samples/sec) and data from a WAV file [9].
- `mingus` is an advanced, cross-platform music theory and notation package for Python with MIDI file and playback support [5].
- `mingus.containers.Bar()` groups notes horizontally [5].
- `mingus.containers.Track()` is a simple data structure to store Bars [5].
- `mingus.extra.lilypond` module provides some methods to help you generate files in the LilyPond format [5] [2].
- `mingus.extra.lilypond.from_Track(track)` Process a Track object and return the LilyPond equivalent in a string [5].

Appendix B Python Codes

```python
#!/usr/bin/env python
# coding: utf-8

# # Part I

# In[1]:


# import packages
import numpy as np
import simpleaudio as sa
import matplotlib.pyplot as plt
from scipy import signal # built-in spectrogram for reference
from matplotlib.animation import FuncAnimation


# In[176]:
```

```
get_ipython().run_line_magic('matplotlib', 'notebook')
get_ipython().run_line_magic('matplotlib', 'notebook')
#%matplotlib inline


# In[233]:


# load handel
y = np.load('y.npy')
y *= 32767/max(abs(y))
y = y.astype(np.int16)

# play handel
play_obj = sa.play_buffer(y, 1, 2, 8000)

# plot handel: amplitude vs time
Fs = np.load('Fs.npy') # rescale factor
n = y.size # number of grid points
t = np.arange(1, n+1)/Fs # rescale frame to time
plt.figure(1)
plt.plot(t, y)
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
#plt.title('Sound amplitude of Handel')
plt.show()


# In[234]:


# Fourier modes
T = t[-1] # period of our "periodic" data, use n since there's no point t = 0.
f = (1.0/T)*np.concatenate((np.arange(0, n/2), np.arange(-(n-1)/2, 0))) # frequency
fs = np.fft.fftshift(f)

# Fourier Transform
yft = np.fft.fft(y)
yfts = np.fft.fftshift(yft)
plt.figure(2)
plt.plot(fs, abs(yfts))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
#plt.title('Fourier Transform of Handel')
plt.show()


# In[129]:


# Gabor Transform
def gabor_transform(data, frames, window_func, window_num = 20, pplot = False, ffilter = False):

    '''
```

```
This function performs Gabor Transform on the data and return horizontal axis, vertical axis and th
data: time series
frames: Number of frames per second
window_num: number of window specified
plot: plotting the process (default = False)
window_func: function for windowing the data
ffilter: filter out secondary peaks (overtones) (default = False)
'''

# time
n = data.size # number of grid points
t = np.arange(1, n+1)/frames # rescale frame to time
tmax = t[-1]
tmin = 0
tslide = np.linspace(tmin, tmax, window_num) # window slide
n2 = int((n+1)/2) # number of absolute points
spc2 = np.zeros((window_num, int((n+1)/2))) # spectrogram with absolute frequency

# vertical axis
f = (1.0/tmax)*np.concatenate((np.arange(0, n/2), np.arange(-(n-1)/2, 0))) # frequency
fs = np.fft.fftshift(f)
fs2 = fs[(n2-1)::]

if pplot:
    fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, ncols=1)
    plt.show()

for j in range(window_num):

    tnow = tslide[j]
    # filter
    g = window_func(t, tnow)

    # Gabor Transform
    yg = data * g
    ygt = np.fft.fft(yg)
    ygts = np.fft.fftshift(ygt)

    # filter
    if ffilter:
        idx_peak = ygts.argmax()
        fs0 = fs[idx_peak]
        filter_width = 10
        filter_func = np.exp( -(fs - fs0)**2 / (2*filter_width**2) )
        ygts = ygts * filter_func

    # save the frequency distribution in spectrogram
    ygtsflip = np.flip(ygts)
    spc2[j] = abs(ygts[0:n2] + ygtsflip[0:n2])
    spc2[j, n2-1] = spc2[j, n2-1]/2 # eliminate counting twice
    spc2[j] = np.flip(spc2[j])

    if pplot:
        fig1, = ax1.plot(t, data)
```

```
            fig2, = ax2.plot(t, yg)
            fig3, = ax3.plot(fs, abs(ygts))
            plt.pause(0.2)

    spc2 = spc2.transpose()

    return tslide, fs2, spc2


# In[238]:


# window functions
width = 0.2 # window width
gaussian = lambda x, x0: np.exp( -(x - x0)**2 / (2*width**2) ) # Gaussian
mexican = lambda x, x0: (1 - (x - x0)**2 / (2*width**2) )*np.exp( -(x - x0)**2 / (2*width**2) ); # Mexi
shannon = lambda x, x0: 1 * (abs(x - x0) <= width); # Shannon function (step)


# In[242]:


# plot three windows
x0 = 0
x = np.linspace(-1,1,100)
plt.figure(100)
plt.plot(x, gaussian(x, x0))
plt.xlabel("$\\tau$")
plt.ylabel("Amplitude")
plt.show()
plt.figure(101)
plt.plot(x, mexican(x, x0))
plt.xlabel("$\\tau$")
plt.ylabel("Amplitude")
plt.show()
plt.figure(102)
plt.plot(x, shannon(x, x0))
plt.xlabel("$\\tau$")
plt.ylabel("Amplitude")
plt.show()


# In[228]:


# gaussian wavelet
tg, fg, spc = gabor_transform(y, Fs, gaussian, window_num = 200)
plt.figure(4)
plt.pcolormesh(tg, fg, spc)
#plt.title('Spectrogram of Handel')
plt.ylabel('Frequency')
plt.xlabel('Time (s)')
plt.colorbar()
plt.show()
```

```
# In[229]:


# mexican hat wavelet
tg, fg, spc = gabor_transform(y, Fs, mexican, window_num = 200)
plt.figure(5)
plt.pcolormesh(tg, fg, spc)
#plt.title('Spectrogram of Handel')
plt.ylabel('Frequency')
plt.xlabel('Time (s)')
plt.colorbar()
plt.show()


# In[230]:


# shannon (step-like) wavelet
tg, fg, spc = gabor_transform(y, Fs, shannon, window_num = 200)
plt.figure(6)
plt.pcolormesh(tg, fg, spc)
#plt.title('Spectrogram of Handel')
plt.ylabel('Frequency')
plt.xlabel('Time (s)')
plt.colorbar()
plt.show()


# In[84]:


ftest, ttest, Sxxtest = signal.spectrogram(y, Fs)
plt.figure(7)
plt.pcolormesh(ttest, ftest, Sxxtest)
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()


# # Part II

# In[38]:


from scipy.io import wavfile


# In[196]:


# music 1 - piano
tsm1 = 16; # record time in seconds
```

```
Fsm1, ym1 = wavfile.read("music1.wav")
nm1 = ym1.size
tm1 = np.arange(0, nm1)/Fsm1

# plot music 1
plt.figure(11)
plt.plot(tm1, ym1)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()

# play music 1
play_m1 = sa.play_buffer(ym1, 1, 2, sample_rate = Fsm1)


# In[197]:


# music 2 - recorder
tsm2 = 14; # record time in seconds
Fsm2, ym2 = wavfile.read("music2.wav")
nm2 = ym2.size
tm2 = np.arange(0, nm2)/Fsm2

# plot music 2
plt.figure(12)
plt.plot(tm2, ym2)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.show()

# play music 2
play_m2 = sa.play_buffer(ym2, 1, 2, sample_rate = Fsm2)


# In[198]:


# Fourier modes
fm1 = (1.0/tm1[-1])*np.concatenate((np.arange(0, nm1/2), np.arange(-(nm1-1)/2, 0))) # frequency
fsm1 = np.fft.fftshift(fm1)

# Fourier Transform
ym1ft = np.fft.fft(ym1)
ym1fts = np.fft.fftshift(ym1ft)
plt.figure(13)
plt.plot(fsm1, abs(ym1fts))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.xlim((-1000,1000))
plt.show()


# In[199]:
```

```
# Fourier modes
fm2 = (1.0/tm2[-1])*np.concatenate((np.arange(0, nm2/2), np.arange(-(nm2-1)/2, 0))) # frequency
fsm2 = np.fft.fftshift(fm2)

# Fourier Transform
ym2ft = np.fft.fft(ym2)
ym2fts = np.fft.fftshift(ym2ft)
plt.figure(14)
plt.plot(fsm2, abs(ym2fts))
plt.xlabel('Frequency')
plt.ylabel('Amplitude')
plt.xlim((-3000,3000))
plt.show()


# In[123]:


# window functions
width = 0.15 # window width
gaussian = lambda x, x0: np.exp( -(x - x0)**2 / (2*width**2) ) # Gaussian


# In[204]:


# gaussian wavelet w/ filtering overtones
# music 1
tg, fg, spc = gabor_transform(ym1, Fsm1, gaussian, window_num = 160, ffilter = True)
# plot spectrogram of music 1
plt.figure(15)
plt.pcolormesh(tg, fg, spc)
plt.ylim((0, 600))
#plt.title('Spectrogram of Music 1 w/o overtones')
plt.ylabel('Frequency')
plt.xlabel('Time (s)')
plt.colorbar()
plt.show()


# The frequency for three nodes played are about 316 Hz, 282 Hz, 251 Hz, respectively corresponding to

# In[205]:


# gaussian wavelet w/ filtering overtones
# music 2
tg2, fg2, spc2 = gabor_transform(ym2, Fsm2, gaussian, window_num = 140, ffilter = True)
# plot spectrogram of music 2
plt.figure(17)
plt.pcolormesh(tg2, fg2, spc2)
plt.ylim((0, 1600))
```

```
#plt.title('Spectrogram of Music 2 w/o overtones')
plt.ylabel('Frequency')
plt.xlabel('Time (s)')
plt.colorbar()
plt.show()


# The frequency for three nodes played are about 1000 Hz, 880 Hz, 789 Hz, respectively corresponding to

# In[192]:


from mingus.containers import Bar, Track
import mingus.extra.lilypond as lilypond

# music 2

b1 = Bar()
b1 + "B-5"
b1 + "A-5"
b1 + "G-5"
b1 + "A-5"

b2 = Bar()
b2 + "B-5"
b2 + "B-5"
b2 + "B-5"
b2 + None

b3 = Bar()
b3 + "A-5"
b3 + "A-5"
b3 + "A-5"
b3 + None

b4 = Bar()
b4 + "B-5"
b4 + "B-5"
b4 + "B-5"
b4 + None

b5 = Bar()
b5 + "B-5"
b5 + "A-5"
b5 + "G-5"
b5 + "A-5"

b6 = Bar()
b6 + "B-5"
b6 + "B-5"
b6 + "B-5"
b6 + "B-5"

b7 = Bar()
```

```
b7 + "A-5"
b7 + "A-5"
b7 + "B-5"
b7 + "A-5"

b8 = Bar()
b8 + "G-5"

t = Track()
t + b1
t + b2
t + b3
t + b4
t + b5
t + b6
t + b7
t + b8

bar = lilypond.from_Track(t)
#lilypond.to_pdf(bar, "my_first_bar") # failed
# manually save to pdf
f = open('music2score.ly', 'w')
f.write(bar)
f.close()
# then execute 'lilypond -fpdf -o "musicscore" "musicscore.ly"'' in command line


# In[191]:


# music 1

b1p = Bar()
b1p + "E-4"
b1p + "D-4"
b1p + "C-4"
b1p + "D-4"

b2p = Bar()
b2p + "E-4"
b2p + "E-4"
b2p + "E-4"
b2p + None

b3p = Bar()
b3p + "D-4"
b3p + "D-4"
b3p + "D-4"
b3p + None

b4p = Bar()
b4p + "E-4"
b4p + "E-4"
b4p + "E-4"
```

```
b4p + None

b5p = Bar()
b5p + "E-4"
b5p + "D-4"
b5p + "C-4"
b5p + "D-4"

b6p = Bar()
b6p + "E-4"
b6p + "E-4"
b6p + "E-4"
b6p + "E-4"

b7p = Bar()
b7p + "D-4"
b7p + "D-4"
b7p + "E-4"
b7p + "D-4"

b8p = Bar()
b8p + "C-4"

tp = Track()
tp + b1p
tp + b2p
tp + b3p
tp + b4p
tp + b5p
tp + b6p
tp + b7p
tp + b8p

barp = lilypond.from_Track(tp)
#lilypond.to_pdf(bar, "my_first_bar") # failed
# manually save to pdf
f = open('music1score.ly', 'w')
f.write(barp)
f.close()
# then execute 'lilypond -fpdf -o "musicscore" "musicscore.ly"'' in command line
```