AMATH 582 Homework 2

Daniel Burnham (https://github.com/burnhamdr)

February 09, 2020

Abstract

The work presented here is motivated by material covered in AMATH 582 Computational Methods For Data Analysis regarding signal time-frequency analysis. For signals recorded in time it is often desirable to understand not just the frequency content of the entire signal, but also how that frequency content changes over the time course of the signal. This can be addressed through implementation of the Gábor transform which is a method for extracting the frequency spectrum from a signal at sequential time steps. This report will highlight the methods of application for this time-frequency analysis approach through analysis of two different sets of audio recordings of music. The first analyzed audio sample is a 9 second sample of the Hallelujah chorus from Handel's Messiah. With this example audio data different Gábor transform filters will be explored, and their effects on the time-frequency analysis results will be characterized. The second part of the report will focus on results obtained from the time-frequency analysis of two separate recordings of a piano and a recorder performing Marry Had a Little Lamb. From the results of the Gabor transform applied to these audio recordings the musical notes will be extracted. Overall, a key idea illustrated through this work is the trade off between accurately localizing a frequency in time versus resolving the frequency band of the signal for a specific time bin. This time-frequency resolution trade off will be discussed in greater detail throughout the following report.

1 Introduction and Overview

Time-frequency analysis is useful for determining when certain frequencies occur within a signal that evolves in time. Audio signals are an example of such a signal, as the sounds that make up the signal occur at individual frequencies and isolated moments in time. The specific audio signals used in this work are the Hallelujah chorus from Handel's "Messiah", and "Mary had a Little Lamb" performed on the piano and the recorder. To extract information about how the frequency spectrum of an audio recording evolves in time, Gábor transforms are applied to each audio signal utilizing various time bin sizes and filter shapes. Experimenting in this way allowed for the advantages and disadvantages of Gábor transformation time-frequency analysis to be visualized.

Spectrograms are a used to visualize the frequency versus time behavior of the Gábor transformed signal while varying properties of the time filtering window. Additional plots are also generated to demonstrate the action of the time filtering window on the signal at successive time steps. From these visualizations the musical score of the piano and recorder renditions of "Mary Had a Little Lamb" were determined.

2 Theoretical Background

The main concept explored with this work is the implementation of the Gábor transform. The Gábor transform is a modification of the Fourier transform kernel aimed at allowing for frequencies to be localized in time. The Gábor transform kernel is shown in Equation 1(Kutz 2013).

$$g_{t,w}(\tau) = e^{i\omega\tau} g(\tau - t) \tag{1}$$

The Gábor transform is also known as the short-time Fourier transform (STFT) and is thus defined similarly to the FFT as representing a function as a sum of cosines and sines Equation 2(Kutz 2013).

$$G[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{i\omega\tau} d\tau \tag{2}$$

The purpose of the function $g(\tau - t)$ is to establish a window centered at $t = \tau$ outside of which the signal is damped to zero. Shifting this window across the signal in time thus allows for sections of the signal to be analyzed for frequency content. Various window functions can be used. For example, in this work we experiment with the following window functions (Kutz 2013):

$$g(t) = e^{-at^2} \quad \text{(Gaussian)} \tag{3}$$

$$g(t) = (1 - at^2)e^{-\frac{t^2}{2}} \quad \text{(Mexican Hat)} \tag{4}$$

$$g(t) = \begin{cases} 0 & |t| > 1/2 \\ 1 & |t| \leq 1/2 \end{cases} \quad \text{(Shannon)} \tag{5}$$

The Gábor method using these sliding windows trades away accuracy in either the time or frequency domains in order to provide a measurement of both time and frequency resolution simultaneously. In performing this windowed signal filtering, signal components with periods longer than the window will not be represented in the frequency spectrum. This limitation is at the heart of the time-frequency resolution trade off where to localize a frequency one must select a shorter time bin to filter, however, this results in the loss of information on low frequency signal components.

2.1 Spectrogram

A spectrogram is a heat map plot of time versus frequency where the intensity of the indicated heat illustrates the amplitude of the frequency bands at each time interval. The spectrogram is generated by first applying the Gábor transform utilizing a filter of interest across the signal. Then, the frequency spectrum data for each time bin is plotted against the time points where the Gábor filters were centered. This results in a heat map of frequency bands indicating the frequency content of the signal as extracted by the Gábor filter at the time points of interest.

3 Algorithm Implementation and Development

The main steps of the algorithm are as follows:

1. Input the audio file, prepare for analysis
2. Generate desired Gábor filter
3. Plot the signal multiplied by the filter to demonstrate the action of the modified Fourier Transform kernel. Also plot frequency spectrum at sequential time bins.
4. Take the Fourier Transform to generate the frequency spectrum of the signal after Gábor filtering at each time step.
5. Plot the spectrogram to illustrate the frequency content of the signal for each time bin.

4 Computational Results

4.1 Part I: Handel Music Gábor Transforms

See Figure 1, Figure 2, and Figure 3 for examples of applying the Gábor method using a Gaussian, Mexican Hat, and Shannon function filter respectively. More in depth analysis is delineated in the attached code where different filter widths and different $d\tau$ values are tested to investigate the impacts on the signal spectrogram. The filters shown in Figure 1, Figure 2, and Figure 3 are examples of instances where oversampling is achieved because the widths of the filters allowed for overlap in the signal extracted by each successive filter shift. Oversampling ensures that good resolution in time and frequency is achieved.

The spectrograms for the Gábor Transforms utilizing the filters depicted in Figure 1, Figure 2, and Figure 3 are shown in Figure 4, Figure 5, and Figure 6 respectively. With the $d\tau$ chosen for these figures,

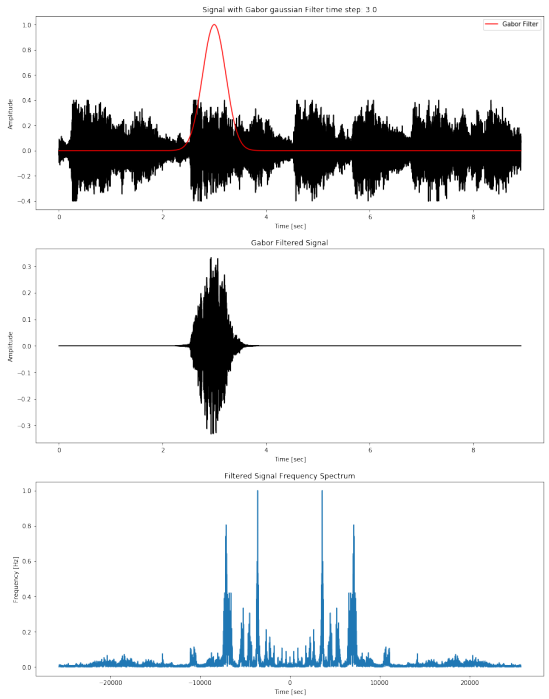

Figure 1: Example of the action of the Gábor method using a standard Gaussian filter for one time bin ($d\tau = 0.1$) within the signal. The top plot shows the complete signal with the filter overlaid. The middle plot shows the result of applying the filter to the signal. The bottom plot is the frequency spectrum of the filtered signal content.

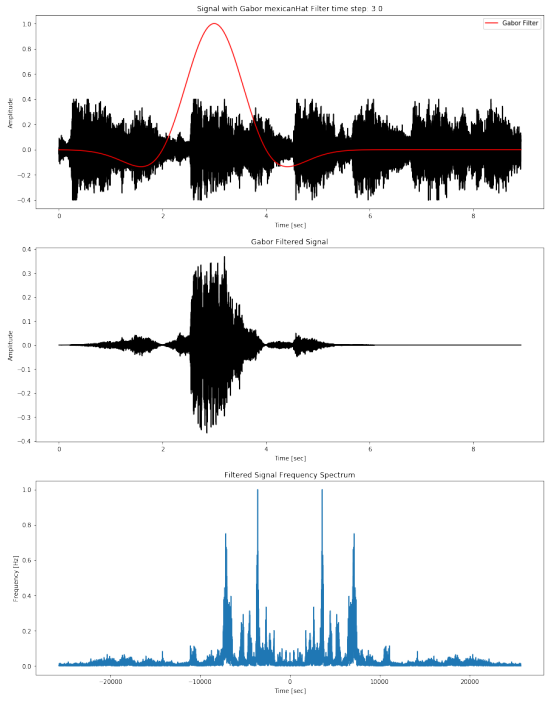

Figure 2: Example of the action of the Gábor method using a Mexican Hat filter for one time bin ($d\tau = 0.1$) within the signal. The top plot shows the complete signal with the filter overlaid. The middle plot shows the result of applying the filter to the signal. The bottom plot is the frequency spectrum of the filtered signal content.

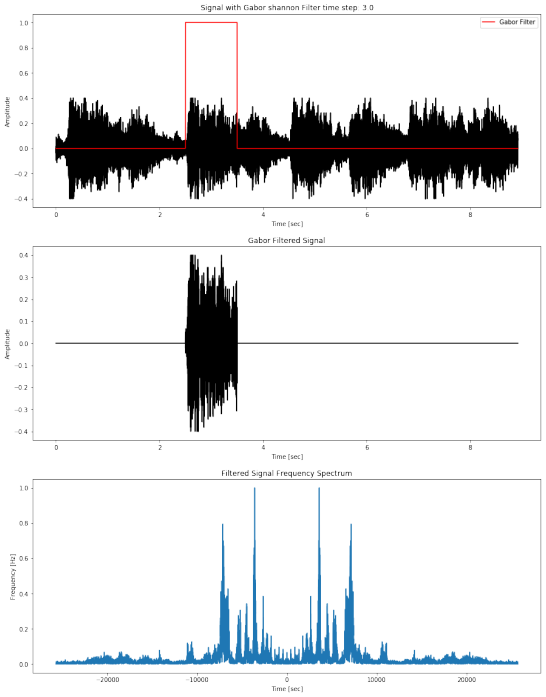

Figure 3: Example of the action of the Gábor method using a Shannon filter for one time bin ($d\tau = 0.1$) within the signal. The top plot shows the complete signal with the filter overlaid. The middle plot shows the result of applying the filter to the signal. The bottom plot is the frequency spectrum of the filtered signal content.

blurring along the time axis can be observed in both the Gaussian and Mexican Hat filter instances. In contrast, the Shannon filter seems to result in a stuttering effect across time most likely caused by the sharpness of the filter boundary. In further investigation carried out in the source code (Appendix B) blurring of the frequency bands in time is shown to be exacerbated by filter width, and blurring of the frequency bands across frequencies is created with a sufficiently narrow filter width. This is consistent with expectations based on the theoretical understanding of the Gábor method described earlier. If the filtering samples large sections in time, the frequency resolution will be good because longer wavelength signals can be represented in the frequency spectrum of the sample extracted by the filter. However, the temporal resolution will be poor because the time bin is large thus making it difficult to determine when the individual frequencies occur in time.

4.2 Part II: Piano and Recorder Music Gábor Transforms

The Gábor Method for time-frequency analysis was performed similarly to Part I. A narrow Gaussian filter with a time bin width $d\tau = 0.1$ was used for the analysis. The spectrograms for the recorder (Figure 7) and piano (Figure 8) music samples of "Mary Had a Little Lamb" indicate sharp bands at the frequencies of the individual notes played. Through detection of the maximum frequency signal in each time bin and converting from wave number to frequency in Hz ($\omega = 2\pi f$), the notes in each musical score were detected and plotted in Figure 9 and Figure 10. Overtones are faintly evident in the piano rendition spectrogram around 250 Hz.

5 Summary and Conclusions

In Part I the Gábor method for performing time-frequency analysis was successfully employed and many cases were explored to illustrate the trade off between time and frequency resolution, and the qualitative

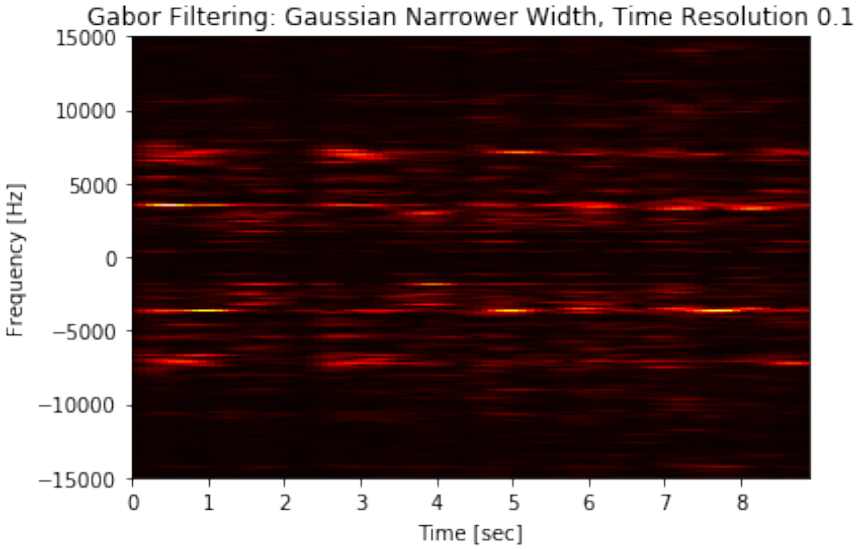

Figure 4: Spectrogram illustrating the frequency content of the signal for each time bin ($d\tau = 0.1$) utilizing a Gaussian filter. The intensity of the heat map represents the scale of the amplitude of the frequency bands.

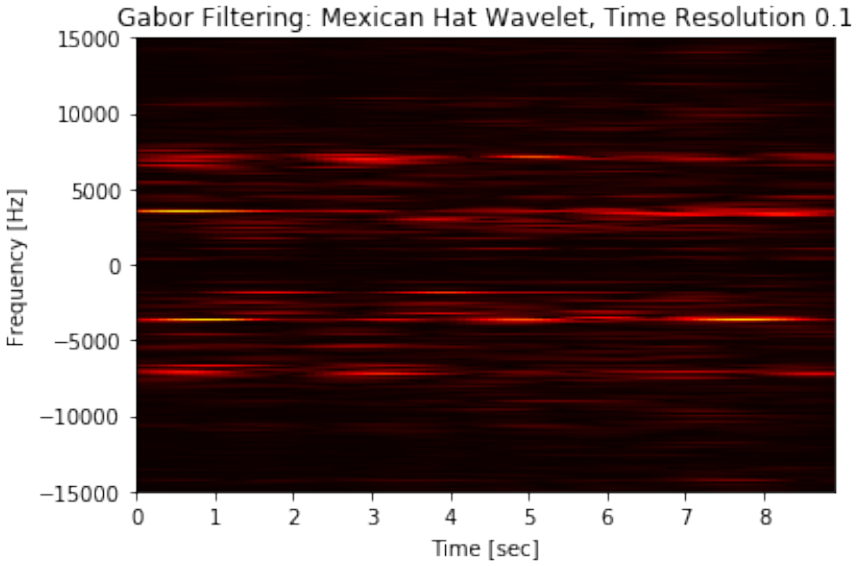

Figure 5: Spectrogram illustrating the frequency content of the signal for each time bin ($d\tau = 0.1$) utilizing a Mexican Hat filter. The intensity of the heat map represents the scale of the amplitude of the frequency bands.

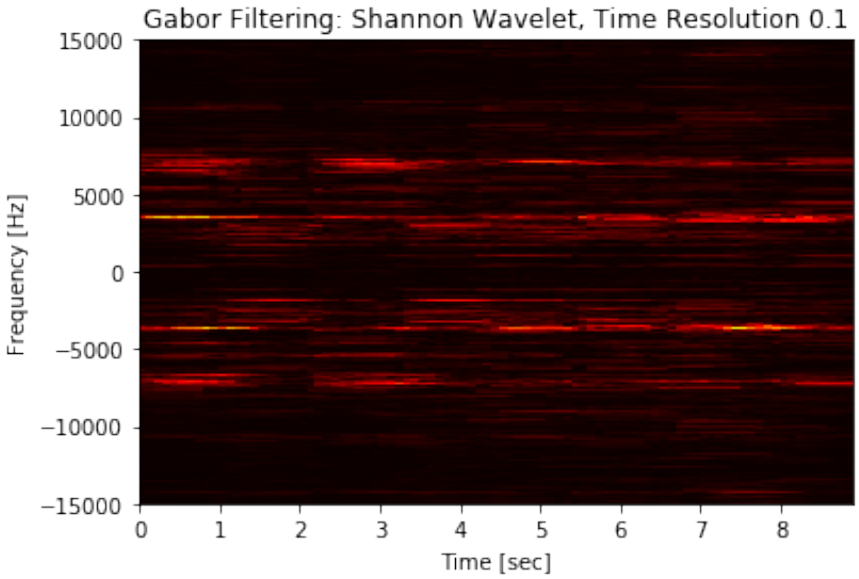

Figure 6: Spectrogram illustrating the frequency content of the signal for each time bin ($d\tau = 0.1$) utilizing a Shannon filter. The intensity of the heat map represents the scale of the amplitude of the frequency bands.

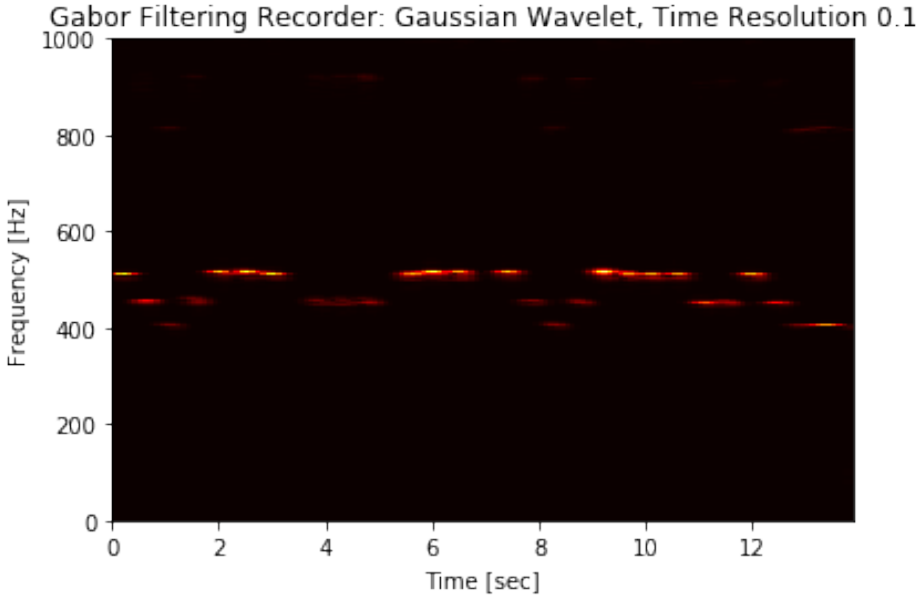

Figure 7: Spectrogram illustrating the frequency content of the recorder rendition for each time bin ($d\tau = 0.1$) utilizing a Gaussian filter. The intensity of the heat map represents the scale of the amplitude of the frequency bands.

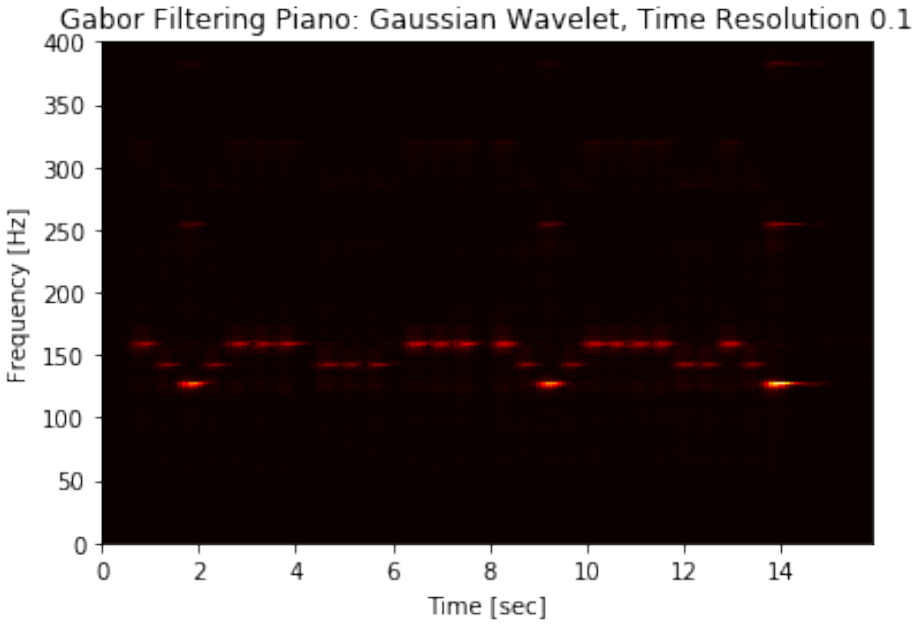

Figure 8: Spectrogram illustrating the frequency content of the piano rendition for each time bin ($d\tau = 0.1$) utilizing a Gaussian filter. The intensity of the heat map represents the scale of the amplitude of the frequency bands.

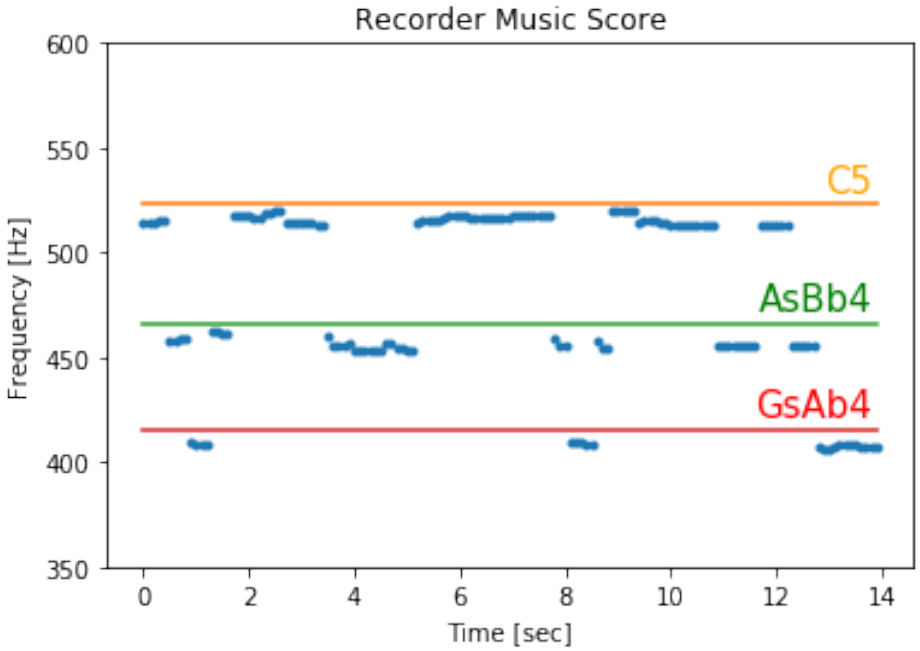

Figure 9: Musical score of the recorder rendition of "Mary Had a Little Lamb"

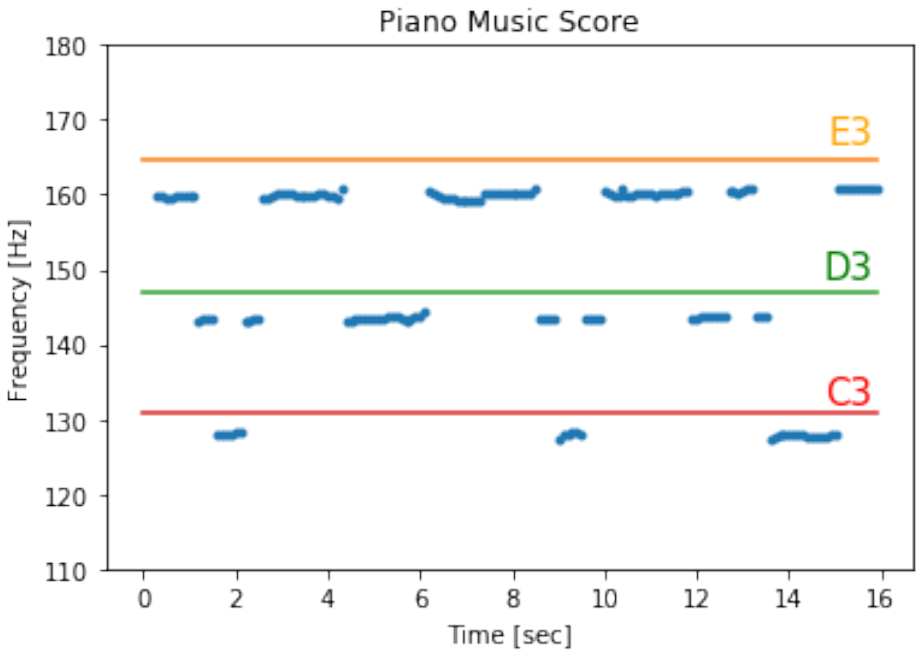

Figure 10: Musical score of the piano rendition of "Mary Had a Little Lamb"

characteristics of different filter shapes. Blurring in the time domain was generated by under-sampling due to a large filter window. Blurring in the frequency domain was observed when the filter window was sufficiently small as to exclude many frequencies from detection based on wavelength.

In Part II the methods practiced in Part I were implemented to determine the musical score of "Mary Had a Little Lamb" played both on the piano and on the recorder. The Gábor method for time-frequency analysis using a narrow Gaussian filter successfully produced spectrograms illustrating bands consistent with an expected musical score. These bands were translated into musical notes by converting the wave number of the maximum frequency signature in each time bin to a temporal frequency.

References

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

Appendix A Python Functions

- `matplotlib.pyplot.pcolormesh(*args, alpha=None, norm=None, cmap=None, vmin=None, vmax=None, shading='flat', antialiased=False, data=None, **kwargs)[source]` Create a pseudocolor plot with a non-regular rectangular grid.
- `numpy.fft.fft2(a, s=None, axes=(-2, -1), norm=None)` Compute the 2-dimensional discrete Fourier Transform. This function computes the n-dimensional discrete Fourier Transform over any axes in an M-dimensional array by means of the Fast Fourier Transform (FFT). By default, the transform is computed over the last two axes of the input array, i.e., a 2-dimensional FFT.
- `numpy.fft.fftshift(x, axes=None)` returns the Shift the zero-frequency component to the center of the spectrum. This function swaps half-spaces for all axes listed (defaults to all). Note that y[0] is the Nyquist component only if len(x) is even.

- `numpy.argmin(a, axis=None, out=None)` Returns the indices of the minimum values along an axis.
- `numpy.argmax(a, axis=None, out=None)` Returns the indices of the maximum values along an axis.

Appendix B Python Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Feb  9 19:05:56 2020

@author: danielburnham
"""

Import Statements
from scipy.io import loadmat
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#functions
def gaborFilter(v, Fs, tres, plot_count, w, ftype='gaussian'):
    '''The Gabor method, or short-time Fourier transform, trades away some measure of
    accuracy in both the time and frequency domains in order to give both time and frequency
    resolution simultaneously.

    This function takes in a two dimensional signal, performs multi-resolution analysis of
    the signal with Gabor filtering for showing signal frequency content over time.

    INPUTS:
    v: numpy.ndarray of signal to analyze.
    Fs: scalar value indicating signal sampling rate. Used to determine time scale for analysis
    ftype: string indicating type of filter to use for the Gabor method
    w: scalar value for scaling the filter width
    tres: scalar value indicating the time resolution to use for filtering.
    plot_count: scalar value indicating number of filtered signal plots to output

    OUTPUTS:
    Sgt_spec: numpy.ndarray of fourier transformed and shifted data in each time bin
    tslide: numpy.ndarray of the time bins used in analysis
    ks: numpy.ndarray of shifted wave numbers

    Plots:
    Plots showing the Gabor filtered signal at time points across the range of time bins
    '''
    isOdd = (len(v) % 2 == 1)# check if there is an odd number of entries
    t2 = (np.arange(0,len(v) + 1)) / Fs
    t = t2[0:len(v)]

    L = (len(v) - 1) / Fs # frequency domain
    n = len(v); # Fourier modes

    tslide = np.arange(0, L, tres)
```

```
    if isOdd: #if there is an odd number of entries, ignore the last data value
        #construct wave numbers
        k = np.concatenate([np.arange(0, (n / 2) - 1), np.arange(-n / 2, -1)])#fourier coeffs list acco
        k = k * (2 * np.pi / L)#convert to a 2 pi periodic domain by rescaling
        ks = np.fft.fftshift(k)#k shifted to plot frequency domain as a funciton of wave number
        Sgt_spec = np.zeros((len(tslide), len(v) - 1))#creates empty matrix to store fft for each windo
    else:
        k = np.concatenate([np.arange(0, (n / 2)), np.arange(-n / 2, 0)])#fourier coeffs list accountin
        k = k * (2 * np.pi / L)#convert to a 2 pi periodic domain by rescaling
        ks = np.fft.fftshift(k)#k shifted to plot frequency domain as a funciton of wave number
        Sgt_spec = np.zeros((len(tslide), len(v)))#creates empty matrix to store fft for each window of

    vt = np.transpose(v)

    plot_num = len(tslide) // plot_count #number of plots to generate across the range of time bins


    for j in range(0, len(tslide)):
        if ftype == 'gaussian':
            g = np.exp(-w*(t - tslide[j])**2) #Gaussian Wavelet
        elif ftype == 'mexicanHat':
            g = (1 - w*(t - tslide[j])**2)*np.exp(-(t - tslide[j])**2) #Mexican hat
        elif ftype == 'shannon':
            g = w*abs(t - tslide[j]) <= 0.5 #Shannon function
        else:
            g = np.exp(-2 * (t - tslide[j])**2); #Gabor filter shifted to center at tslide[j]

        Sg = np.transpose(g * vt) #apply filter to signal
        if isOdd: #if there is an odd number of entries, ignore the last data value
            Sgt = np.fft.fft2(Sg[0:len(v) - 1]) #calculate fourier transform of filtered signal
        else:
            Sgt = np.fft.fft2(Sg[0:len(v)]) #calculate fourier transform of filtered signal
        Sgt_spec[j, :] = np.transpose(abs(np.fft.fftshift(Sgt))) #save shifted fourier transform for sp
        if (j % plot_num == 0):
            f, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(15,20))
            ax1.plot(t, v, color='black')
            ax1.plot(t, np.transpose(g), color='red', label='Gabor Filter')
            ax1.set_title('Signal with Gabor '+ ftype +' Filter time step: ' + str(tslide[j]))
            ax1.set_ylabel('Amplitude')
            ax1.set_xlabel('Time [sec]')
            ax1.legend(loc='upper right')
            ax2.plot(t, Sg, color='black')
            ax2.set_title('Gabor Filtered Signal')
            ax2.set_ylabel('Amplitude')
            ax2.set_xlabel('Time [sec]')
            ax3.plot(ks, abs(np.fft.fftshift(Sgt))/max(abs(np.fft.fftshift(Sgt))))
            ax3.set_title('Filtered Signal Frequency Spectrum')
            ax3.set_ylabel('Frequency [Hz]')
            ax3.set_xlabel('Time [sec]')

    return Sgt_spec, tslide, ks

def spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title):
    '''spectroPlot generates a spectrogram plot of Gabor filtered two dimensional signal data
```

```
    INPUT:
    Sgt_spec: Sgt_spec: numpy.ndarray of fourier transformed and shifted data in each time bin
    tslide: numpy.ndarray of the time bins used in analysis
    ks: numpy.ndarray of shifted wave numbers
    xlim: tuple indicating x axis minimum and maximum (min, max)
    ylim: tuple indicating y axis minimum and maximum (min, max)'''

    f, ax = plt.subplots()
    ax.pcolormesh(tslide, ks, np.transpose(Sgt_spec), cmap = 'hot')
    ax.set_ylabel('Frequency [Hz]')
    ax.set_xlabel('Time [sec]')
    ax.set_title(title)
    ax.set(xlim = xlim, ylim = ylim)

def find_nearest(array, value):
    '''find_nearest takes in an array and a value and finds the index of the value in the array
    of which the passed value is closest to.

    INPUT:
    array: numpy.ndarray of values to search within for the entry closest to the value of interest
    value: scalar value to compare to each entry of the array

    OUTPUT:
    idx: scalar value indicating the index of the array that contains the value closest to the value of
    '''
    array = np.asarray(array)
    idx = (np.abs(array - value)).argmin()
    return idx

#Data matrix contains 2 dimensional handel music sample
y_ = loadmat('y.mat')#load MATLAB .mat file into python as a dictionary
y = y_['y']#raw 'handel' music data matrix

Fs_ = loadmat('Fs.mat')#load MATLAB .mat file into python as a dictionary
Fs = Fs_['Fs']#raw data

r, c = y.shape

v = y / 2
t = (np.arange(0,len(v))) / Fs[0]#create time array for plotting

#plot of 'handel' music file from MATLAB
fig, ax = plt.subplots()
ax.plot(t, v)#plot signal in time
ax.set_xlabel('Time [sec]')
ax.set_ylabel('Amplitude')
ax.set_title('handel MATLAB music signal')

'''The Gabor method, or short-time Fourier transform, trades away some measure of
accuracy in both the time and frequency domains in order to give both time and frequency
resolution simultaneously.

In practice, the Gabor transform is computed by discretizing the time and
```

```
frequency domain. Thus a discrete version of the transform needs to
be considered.

Kutz, J. (2013). Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data
Oxford: Oxford University Press.'''

#Try Gaussian filter, time sampling 0.1
tres = 0.1
plot_count = 6
w = 1
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='gaussian')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: Gaussian, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try Gaussian filter, with coarser time sampling 1
tres = 1
plot_count = 6
w = 1
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='gaussian')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: Gaussian with Coarser Time Resolution 1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try Narrower Gaussian filter, time sampling 0.1
tres = 0.1
plot_count = 6
w = 10
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='gaussian')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: Gaussian Narrower Width, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try wider Gaussian filter, time sampling 0.1
tres = 0.1
plot_count = 6
w = 0.1
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='gaussian')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: Broader Gaussian, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try Mexican Hat wavelet, time sampling 0.1
tres = 0.1
plot_count = 6
w = 1
```

```
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='mexicanHat')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: Mexican Hat Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try bigger Mexican Hat wavelet, time sampling 0.1
tres = 0.1
plot_count = 6
w = 10
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='mexicanHat')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: deeper Mexican Hat Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try smaller Mexican Hat wavelet, time sampling 0.1

tres = 0.1
plot_count = 6
w = 0.1
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='mexicanHat')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: shallower Mexican Hat Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try Shannon wavelet, time sampling 0.1

tres = 0.1
plot_count = 6
w = 1
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='shannon')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: Shannon Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)

#Try narrower Shannon wavelet, time sampling 0.1

tres = 0.1
plot_count = 6
w = 10
[Sgt_spec, tslide, ks] = gaborFilter(v, Fs[0], tres, plot_count, w, ftype='shannon')

xlim = (None, None)
ylim = (-15000, 15000)
title = 'Gabor Filtering: narrow Shannon Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec, tslide, ks, xlim, ylim, title)
```

```
##PART TWO
tr_piano = 16 # record time in seconds
y_ = loadmat('music1.mat')#load MATLAB .mat file into python as a dictionary
yp = y_['y']#raw piano music data matrix
Fsp = len(yp) / tr_piano
tp = np.arange(0, len(yp)) / Fsp

#plot of Mary had a little lamb piano wav file
fig, ax = plt.subplots()
ax.plot(tp, yp)
ax.set_xlabel('Time [sec]')
ax.set_ylabel('Amplitude')
ax.set_title('Mary had a little lamb (piano)')

tr_rec = 14 # record time in seconds
y_ = loadmat('music2.mat')#load MATLAB .mat file into python as a dictionary
yr = y_['y']#raw recorder music data matrix
Fsr = len(yr) / tr_rec
tr = np.arange(0, len(yr)) / Fsr

#plot of Mary had a little lamb recorder wav file
fig, ax = plt.subplots()
ax.plot(tr, yr)
ax.set_xlabel('Time [sec]')
ax.set_ylabel('Amplitude')
ax.set_title('Mary had a little lamb (recorder)')

#Try smaller gaussian wavelet, time sampling 0.5, on recorder music
tres = 0.1
plot_count = 6
w = 20

[Sgt_spec_r, tslide_r, ks_r] = gaborFilter(yr, Fsr, tres, plot_count, w, ftype='gaussian')

xlim = (None, None)
ylim = (0, 1000)
title = 'Gabor Filtering Recorder: Gaussian Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec_r, tslide_r, ks_r / (4*np.pi), xlim, ylim, title)

#Try smaller gaussian wavelet, time sampling 0.1 on piano music
tres = 0.1
plot_count = 6
w = 30

[Sgt_spec_p, tslide_p, ks_p] = gaborFilter(yp, Fsp, tres, plot_count, w, ftype='gaussian')

xlim = (None, None)
ylim = (0, 400)
title = 'Gabor Filtering Piano: Gaussian Wavelet, Time Resolution 0.1'
spectroPlot(Sgt_spec_p, tslide_p, ks_p / (4*np.pi), xlim, ylim, title)

#Determine score of recorder music
names = ['note', 'frequency', 'wavelength']#names for the columns of the dataframe defined as "notes"
#notes dataframe to hold information on the frequencies and names of musical notes
```

```
notes = pd.read_excel('musicNotes.xlsx', index_col=None, header=None, names=names)


score = []#array to hold the names of the detected notes or the musical score
freqs = []#array to hold the frequencies in Hz of the detected notes

#iterate over each time bin and determine the frequency in Hz of the note played
for i in range(0, len(tslide_r)):

    max_ind = np.argmax(Sgt_spec_r[i, :])#find max entry of the frequency spectrum for the time bin
    wave_num = ks_r[max_ind]#find the associated wave number

    freq = abs(wave_num) / (4*np.pi)#calculate the frequency in Hz of the wave number
    note_ind = find_nearest(notes['frequency'], freq)#find the index of the that frequency in the notes

    note = notes.loc[note_ind, 'note']#find the note associated with the frequency of interest
    score.append(note)#add note to score
    freqs.append(freq)#add frequency to list of frequencies (for score plotting)

#plot the musical score
f, ax = plt.subplots()
ax.plot(tslide_r, freqs, ls='', marker='.')#plot the max frequency detected in each time bin
ax.plot(tslide_r, notes.loc[notes['note'] == 'C5', 'frequency'].values*np.ones(len(tslide_r))) #plot li
ax.plot(tslide_r, notes.loc[notes['note'] == 'AsBb4', 'frequency'].values*np.ones(len(tslide_r)))#plot
ax.plot(tslide_r, notes.loc[notes['note'] == 'GsAb4', 'frequency'].values*np.ones(len(tslide_r)))#plot
#annotate line of dominant note
ax.text(0.95, 0.7, 'C5',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='orange', fontsize=15)
#annotate line of dominant note
ax.text(0.95, 0.475, 'AsBb4',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='green', fontsize=15)
#annotate line of dominant note
ax.text(0.95, 0.275, 'GsAb4',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='red', fontsize=15)
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [sec]')
title = 'Recorder Music Score'
ax.set_title(title)
xlim = (None, None)
ylim = (350, 600)
ax.set(xlim = xlim, ylim = ylim)

#Determine score of piano music
score = []#array to hold the names of the detected notes or the musical score
freqs = []#array to hold the frequencies in Hz of the detected notes

#iterate over each time bin and determine the frequency in Hz of the note played
for i in range(0, len(tslide_p)):
```

```
    max_ind = np.argmax(Sgt_spec_p[i, :])#find max entry of the frequency spectrum for the time bin
    wave_num = ks_p[max_ind]#find the associated wave number

    freq = abs(wave_num) / (4*np.pi)#calculate the frequency in Hz of the wave number
    note_ind = find_nearest(notes['frequency'], freq)#find the index of the that frequency in the notes

    note = notes.loc[note_ind, 'note']#find the note associated with the frequency of interest
    score.append(note)#add note to score
    freqs.append(freq)#add frequency to list of frequencies (for score plotting)

#plot the musical score
f, ax = plt.subplots()
ax.plot(tslide_p, freqs, ls='', marker='.')#plot the max frequency detected in each time bin
ax.plot(tslide_p, notes.loc[notes['note'] == 'E3', 'frequency'].values*np.ones(len(tslide_p))) #plot li
ax.plot(tslide_p, notes.loc[notes['note'] == 'D3', 'frequency'].values*np.ones(len(tslide_p)))#plot lin
ax.plot(tslide_p, notes.loc[notes['note'] == 'C3', 'frequency'].values*np.ones(len(tslide_p)))#plot lin
#annotate line of dominant note
ax.text(0.95, 0.8, 'E3',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='orange', fontsize=15)
#annotate line of dominant note
ax.text(0.95, 0.54, 'D3',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='green', fontsize=15)
#annotate line of dominant note
ax.text(0.95, 0.3, 'C3',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='red', fontsize=15)
ax.set_ylabel('Frequency [Hz]')
ax.set_xlabel('Time [sec]')
title = 'Piano Music Score'
ax.set_title(title)
xlim = (None, None)
ylim = (110, 180)
ax.set(xlim = xlim, ylim = ylim)
```