Time-Frequency Analysis of Handel and Comparison of Piano and Recorder Compositions

Wyatt Scherer – *University of Washington*
AMATH 582 HW 2, Due 2/07/2020
GitHub Repository: https://github.com/wcscherer/DATA_ANALYSIS-AMATH582

ABSTRACT
Part one of this paper analyzes a nine second section of Handel's *Messiah Hallelujah Chorus*. Three Gábor filters are developed and the characteristics of these filters are compared by analyzing the frequency content of Handel's *Messiah*. Part two of this analysis compares recordings of *Mary Had a Little Lamb* performed on a piano and on a recorder. The timbre of the two instruments are compared by analyzing the frequencies extracted from each piece. The same Gábor filters are also used to recreate the score of *Mary Had a Little Lamb* by isolating the frequency of each note and the time it was played.

1. Introduction

Meaningful time-frequency analysis requires sampling a given signal over different time partitions of that signal. A common practice is to apply a Fourier transform to a signal to determine the frequency components that make up that signal. Transforming the entire signal can be sufficient if the signal is stationary in time. If the signal changes over time, than the associated frequencies of that signal change over time as well. Determining when certain frequencies occur within the signal requires transforming small sections of the signal. By transforming smaller time windows of the signal, the frequency content over that smaller timeframe can easily be extracted from the data. This process allows for determining the signal's frequency components over a small section of the signal, providing useful data when specific frequencies occur within the signal. The tradeoff; however, is that as the time (frequency) resolution of the partitioning increases, the frequency (time) resolution decreases.

1.1. Part 1: Analyzing Handel's *Messiah*

MATLAB includes a nine second clip of the Hallelujah chorus of Handel's *Messiah*. This clip provides a useful example for building and testing different filters. The performance of these filters on the Handel piece will provide intuition for applying the same techniques and filters on the piano and recorder data in Part 2.

1.2. Part 2: Comparing Piano and Recorder Compositions

Two versions of *Mary had a Little Lamb* are analyzed: one played on the piano and one on the recorder. The filtering techniques developed in Part 1 are used to compare the frequency spectra of the piano and recorder pieces. The number and quality of overtones present (timbre) for each instrument will be compared. Finally, the frequency data of each recording will be filtered over discrete time steps to provide an approximate score of each piece.

2. Technical Basis for Time-Frequency Filtering

Time-frequency filtering requires two main processes: 1) Fourier transformation and 2) signal filtering. Fourier transformations are necessary to extract the frequency information from the signal. Signal filtering using shifting Gábor filters is necessary for selecting portions of the total signal to extract the frequency data of that given partition.

2.1. Fourier Transforms and the FFT Algorithm

A Fourier transform is built on the definition of Fourier series. A Fourier series represents an arbitrary function *f(x)* as an infinite sum of sines and cosines. Applying a Fourier transform to the raw audio signal will provide the associated signal frequency data of the section of the song. Computationally, the Fast Fourier Transform (FFT) algorithm allows for fast computation, $O(N\log(N))$, of Fourier transforms. The key to this speed is that the FFT algorithm discretizes the $x\in[-L, L]$ domain into 2^n points. This means that the associated transformed frequencies, k, are also discretized into 2^n possible frequencies. This process produces periodic solutions of n points on $x\in[-L, L]$ [1]. It is important to note that data operated on by the FFT algorithm must be shifted by half of a period along the transformed domain to be viewed in its mathematically correct position. The magnitude of the wave numbers, k, produced by the transform represent the 'weight' that specific frequency consist of in the transform of *f(x)* [1].

2.2. Gábor Filter Development

For non-stationary signals, a Fourier transform only provides the total frequency content of a given signal over the entire length of the signal. To determine what frequencies occur at a given time, a windowed Fourier transform should be implemented. A windowed Fourier transform takes a segment of the total signal (a window) from t_1 to t_2 and Fourier transforms this window. This filtering provides the signal frequency content that occurs over the window from t_1 to t_2.

Modifying the transformation kernel of the Fourier transform allows for transforming specific windows of the total signal *S(t).* The generic form of these modified filters is a *Gábor Transform* [1].

$$g_{t,\omega}(\tau) = \widetilde{f_g}(t,\omega) = e^{i\omega\tau}g(\tau - t) \quad (1)$$

Here *g(t-τ)* defines the generic transform kernel centered on τ. Generally, the kernel g is both real symmetric and symmetric around τ with a L_2 norm such that $||g(t)||$ = 1 and $||\ g(t\text{-}\tau)||$ = 1. With these characteristics, the *Gábor transform* is defined as

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

Because the Gábor transform extracts the frequencies of a *part* of the total signal defined by *g(t-τ)*, the resolution of the frequency domain depends on the resolution of the time domain.; specifically, the better the time resolution of the Gábor kernel, the worse resolution of the frequency resolution and vise versa The uncertainty in the frequency domain and the time domain follows the Heisenberg uncertainty principle – see Appendix A for further explanation [1]. The inverse of the Gábor transform is defined as

$$f(\tau) = \frac{1}{2\pi}\frac{1}{\|g\|^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\widetilde{f_g}(t,\omega)g(\tau - t)e^{-i\omega\tau}d\omega dt \quad (3)$$

In practice, the Gábor transform is implemented as a discrete transform over the time and frequency domains. The discrete lattice of time and frequency space points are

$$\nu = m\omega_0, \ \ \tau = nt_0 \quad (4)$$

where m and n are integers and ω_0 and t_0 are constants greater than 0 [1]. The discrete transform kernel then becomes

$$\tilde{f}(m,n) = \int_{-\infty}^{\infty} f(t)\tilde{g}_{m,n}(t)dt = (f, g_{m,n}) \quad (5)$$

The Gábor filters are utilized in MATLAB by applying the Gábor kernel with the defined width and center τ to the input signal. Then the filtered signal is transformed using a Fourier transform. This provides the signal frequency components over the filtered signal time domain.

2.3. Gábor Kernel Choices

Three kernels were chosen for testing on the clip from Handel's *Messiah*: a Gaussian kernel, a Ricker kernel (Mexican Hat wavelet), and a Shannon kernel.

2.3.1. Gaussian Kernel

The Gaussian kernel is defined by a width w, and a center point τ:

$$G_k(w,\tau,t) = e^{-\frac{1}{w}(t-\tau)^2} \quad (6)$$

2.3.2. Ricker Kernel

The Ricker kernel, commonly known as the Mexican Hat wavelet, is the second moment of a Gaussian. It is also defined by a width w, and a center point τ*:*

$$R_k(w,\tau,t) = \left(1 - \left(\frac{\tau - t}{w}\right)^2\right)e^{-\left(\frac{t-\tau}{2w}\right)^2} \quad (7)$$

2.3.3. Shannon Kernel

The Shannon kernel is a piecewise function defined by a width w, and a center point τ*:*

$$S_k(w,\tau,t) = \begin{cases} 0 \ where\ t < \tau - w \\ 1 \ where\ \tau - w \le t \le \tau + w \\ 0 \ where\ t > \tau + w \end{cases} \quad (8)$$

3. Algorithm Development

The algorithm for importing and analyzing the signal is similar for analyzing Handel, the piano composition, and the recorder composition. See Appendix A for brief explanations of MATLAB functions used in the analysis.

3.1. Import Signal Data

- Import sound file using the *audioread* function, providing the signal as a vector and provides the sampling rate (Fs)
- Create the timespan vectors of the signal based on the length of the signal, n
- Create the shifted and un-shifted wave number vectors for plotting – use all of the points in the

signal since the signal is not periodic: k = [0:n/2 -n/2:-1]
- Use the ratio of the sampling rate and the number of points in the signal to determine the conversion from wave number to frequency in Hertz: r = (Fs/n)
- The audio signal can now be plotted, transformed using *fft*, and filtered

3.2. Build Gábor Filters

- Build vectors for the widths evaluated for each filter: Gaussian, Ricker, and Shannon
- Build the vectors of the center points for each filter that sample from the signal time domain
- Use the custom *build_filt* function that builds a 3D tensor for each filter kernel over the span of the signal, for each center location, and each width
- *build_filt* calls other custom functions *GG_filter, MHW_filter, and SH_filter* that build Gaussian, Ricker, and Shannon filters respectively as 3D tensors
- Once the filter tensors are created, the tensors are passed to the custom function *filter_data,* which applies the input filter tensor to the input signal that is being filtered
- *filter_data* outputs three separate tensors for the input filter tensor and input signal vector: 1st tensor is the input signal with the applied filter, 2nd tensor is the Fourier transformed filtered signal, 3rd tensor is the shifted Fourier transformed input signal

3.3. Visualize Filtered Signal Data

- *filter_data* provides all the filtered, transformed signal data necessary for final analysis
- build spectrograms of the filtered signal data using *pcolor* to plot the frequency content of each Gábor filtered portion of the audio signal at the time value it was centered at.
- Use spectrograms to compare the signal frequency and time resolutions the different filters provide
- Modify the spectrograms by plotting subsets of the frequency domain to provide better visual representations of the signal
- Test the three filters on Handel's *Messiah* to determine which filter provides the best frequency resolution on the spectrograms. Use the superior filter to recreate the score of *Mary had a Little Lamb* from the filtered piano and recorder audio files.

4. Computational Results

4.1. Handel's *Messiah*

MATLAB includes an approximately 9 second long segment of Handel's *Messiah Hallelujah Chorus.* The raw audio signal data was imported and plotted – see Figure 1. The Fourier transform of the whole signal to determine the total frequency composition of the sound file is shown in Figure 2.

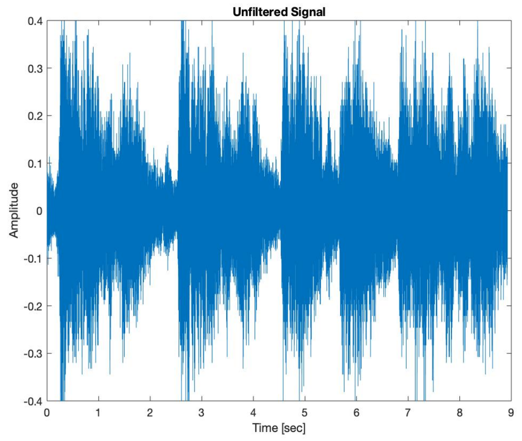

Figure 1: Unfiltered *Hallelujah Chorus* Signal Sample

The *Hallelujah Chorus* includes male and female singers. The frequencies represented in Figure 2 illustrate the peaks corresponding to the lower pitch male frequencies, the higher pitch female frequencies, and regions of overlap. According to the Sound Engineering Academy (Ref. [2]), the male voice fundamental frequency ranges from 100Hz to 900Hz, with harmonics up to 8 kHz. The female voice fundamental frequency ranges from 350 Hz to 3 kHz with harmonics up to 17 kHz [2]. The high peaks represent the most frequent fundamental frequencies sung while the lower peaks represent less frequent frequencies and harmonics of the fundamental notes.

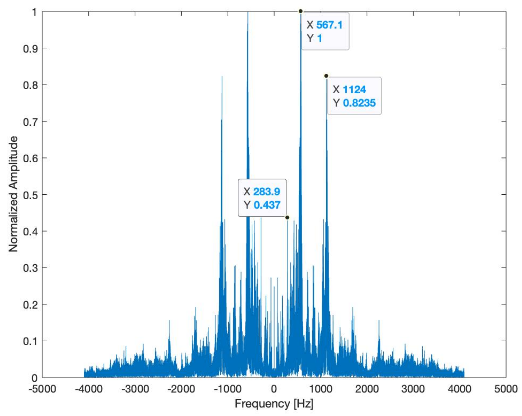

Figure 2: Fourier Transform of total Handel Sample

To illustrate the effect of varying Gábor filter width on frequency vs. time resolution, three spectrograms were produced using a Gaussian filter with widths 0.01, 0.05, and 0.10. The filter was shifted over 90 points from the beginning to the end of the signal, with a Δt of approximately 0.10 seconds. Only the positive frequencies will be plotted since they correspond to physical frequencies.

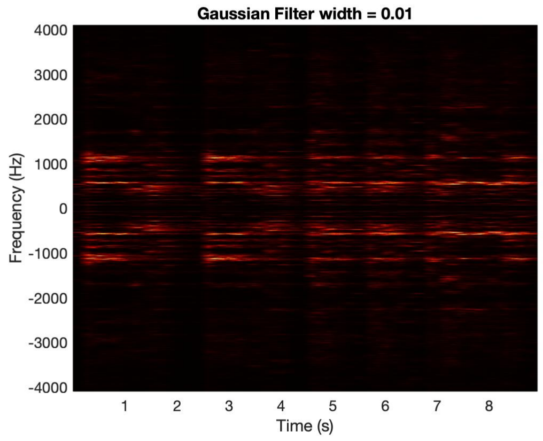

Figure 3: Handel Sample with Gaussian Filter with w = 0.01

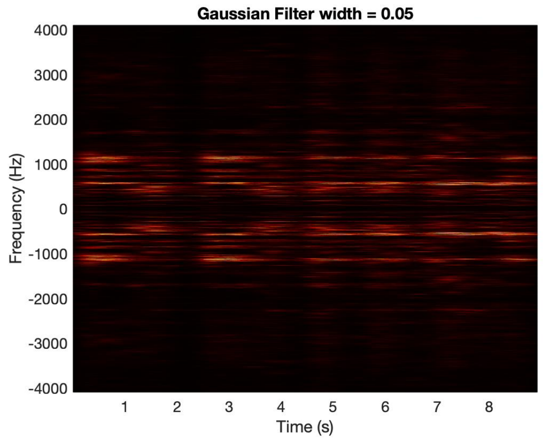

Figure 4: Handel Sample with Gaussian Filter with w = 0.05

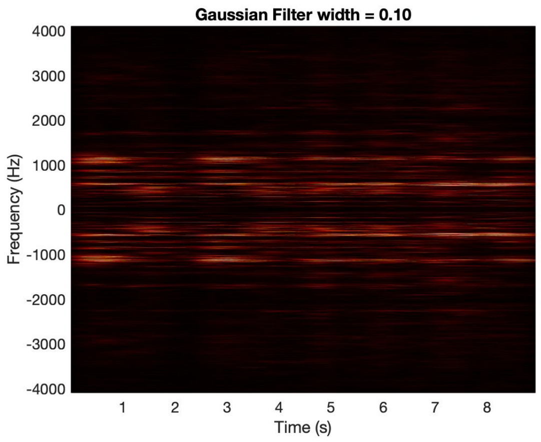

Figure 5: Handel Sample with Gaussian Filter with w = 0.10

The effect on frequency resolution filter width has can clearly be seen by comparing Figure 3 to Figure 5. With a smaller Gaussian filter width, as in Figure 3, the number of visible frequencies is significantly smaller than in Figure 5. However, the visible frequencies have sharper definition in time. In Figure 5, the frequencies appear to be smeared out over time, but more frequencies are visible. As expected, the larger filter width provides more frequencies but less time resolution, whereas a smaller width provides fewer frequencies, but has better time resolution.

Figures 6 and 7 show the effect of different filters on the resolution of spectrograms. Utilizing the same filter width of 0.05, the Ricker and Shannon filters were also shifted over 90 points from the beginning to the end of the Handel sample. These spectrograms can be compared to Figure 4, which was created with a Gaussian filter with a width of 0.05.

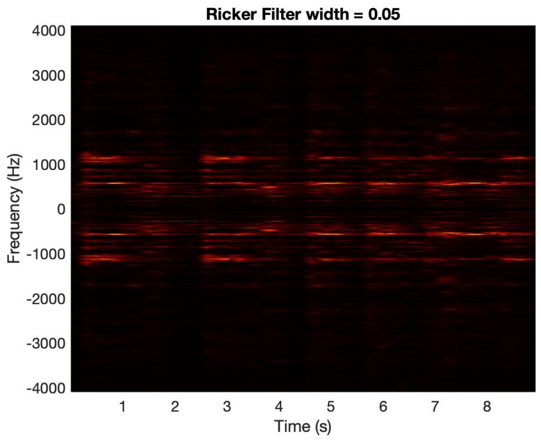

Figure 6: Handel Sample with Ricker Filter with w = 0.05

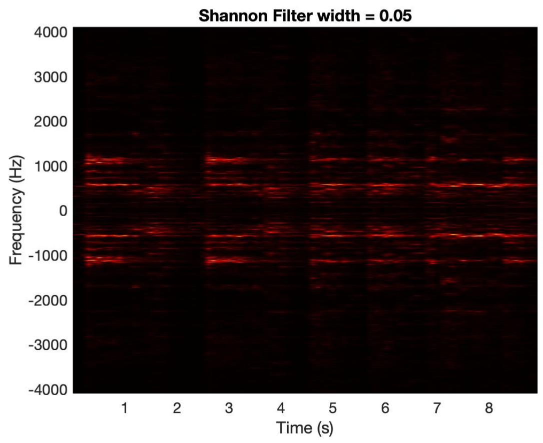

Figure 7: Handel Sample with Shannon Filter with w = 0.05

Both Figure 6 and Figure 7 show better time resolution than the Gaussian filter in Figure 5 when the filter width is the same and the filter is shifted over the same number of points. The Ricker filter in Figure 6 captures more frequencies per a given slice in time, but the Shannon filter in Figure 7 has better resolution in time. The Shannon filter's discrete edges mean that it filters only a signal width of *2w*. Both the Gaussian and Ricker filters extend beyond the width w, which allows them to capture more frequencies. The Ricker filter's inverted peaks to the left and right of the positive center peak mean it can capture more frequencies than a Shannon filter, but weight the center frequencies more sharply than a simple Gaussian filter of the same width.

Based on this analysis, a Shannon filter should be used when high frequency in time resolution of a given signal is required, and a Ricker filter should be used when a broader frequency bandwidth with moderate time resolution is required. The Gaussian filter should be used when a large frequency bandwidth is required or if a small filter width can be used. A small filter width means the filter has to be shifted over more points along the signal to ensure the all the signal is sampled. This creates a larger tensor, and increases the computation time of the analysis. A Ricker or Shannon filter can give better frequency and or time resolution with a larger filter width and less translations.

4.2. Piano Composition of *Mary Had a Little Lamb*

Mary had a Little Lamb is commonly played in the key of C (middle C) and requires only three notes E, D, and C. This means that three frequencies corresponding to E (~330 Hz), D (~294 Hz) and middle C (~262 Hz) should be clearly visible in a spectrogram. The first rendition is played on a Piano.

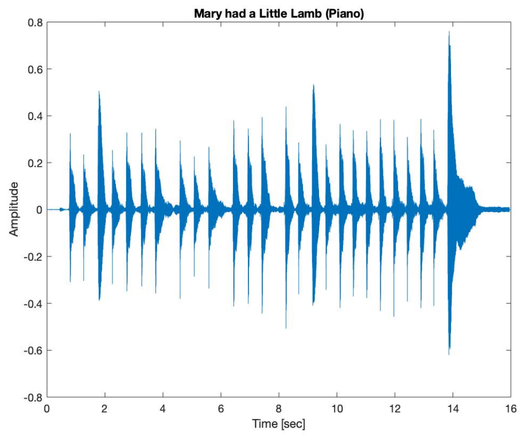

Figure 8: Audio of Mary had a Little Lamb on Piano

Figure 8 shows the signal sample and Figure 9 shows the Fourier transform of the total signal. The piano audio sample is approximately 16 seconds long. Because the Fourier transform transforms the signal into *2n* frequencies, where *n* is the number of data points in the signal (length), there are far more frequencies transformed form the signal than are played in the song. Negative frequencies can be discarded since they are un-physical solutions and frequencies past 3 kHz can be discarded since the song is played in the key of C on middle C. There will be some higher overtones, but they will not be very visible in the spectrograms. Figure 9 will show the frequencies of the main three notes (E, D, and C) along with their higher overtones, which are integer multiples of the fundamental frequency of the note.

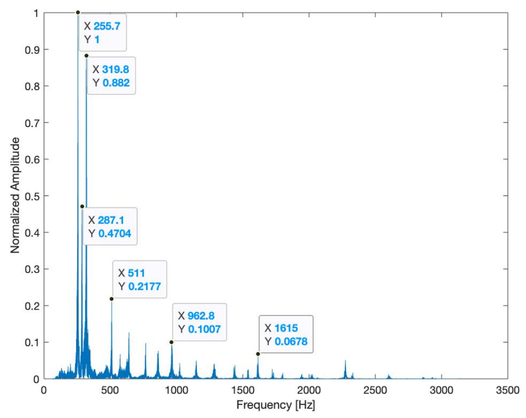

Figure 9: Fourier Transform of the Piano Composition

As can be seen from Figure 9, the three most prominent frequencies are 256 Hz, 287 Hz, and 320 Hz. These frequencies correspond to C, D, and E respectively, although the piano seems to be flat as these frequencies are lower than what the true frequencies should be. Also, some of the overtones of C (511 Hz and 1615 Hz) and E (962.8 Hz) have been identified. The amplitude of the frequencies represents their total weight of the signal that each frequency contributes. From Figure 9, the note C has the largest weight, which corresponds to the signals with the highest amplitude and longest length in time as displayed in Figure 8.

To recreate a score of the Piano piece, a spectrogram using a Shannon kernel Gábor filter was applied. A Shannon kernel was chosen for its superior time resolution properties as identified in Section 4.1. The filter has a width of 0.05 and the shift parameter τ corresponds to 0.10 seconds. The Shannon filter shifts over 160 points along the length of the song. Figure 10 displays the score of the song created from the transformed Shannon filtered data.

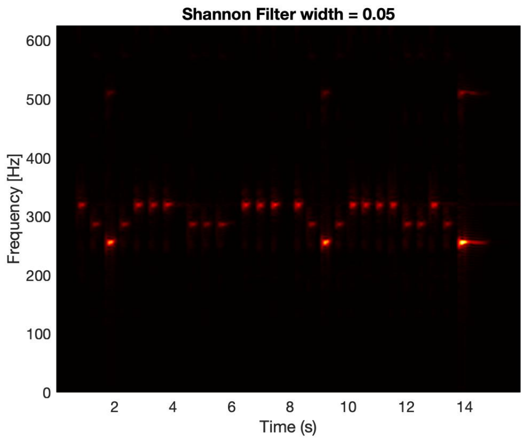

Figure 10: Score of the Piano Composition for *Mary Had a Little Lamb*

Figure 10 clearly shows when in time each note E (~320 Hz), D (~287 Hz), and C (~256 Hz) are played in the audio file.

4.3. Recorder Composition of *Mary Had a Little Lamb*

The same composition of *Mary Had a Little Lamb* was performed on a recorder. This recording was 14 seconds long. The same filtering process used on the piano audio file was used on the recorder audio file. Figure 11 shows the signal sample and Figure 12 shows the Fourier transform of the total signal.

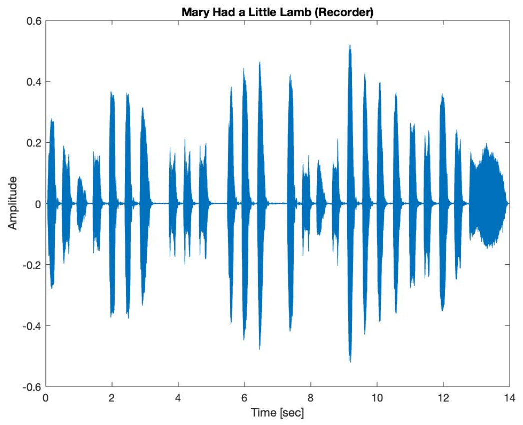

Figure 11: Audio of Mary had a Little Lamb on Recorder

Figure 11 shows that the intensity with which each note was played on the recorded varies more than the piano composition. This variance will result in significantly different weights of the frequencies comprising the piece.

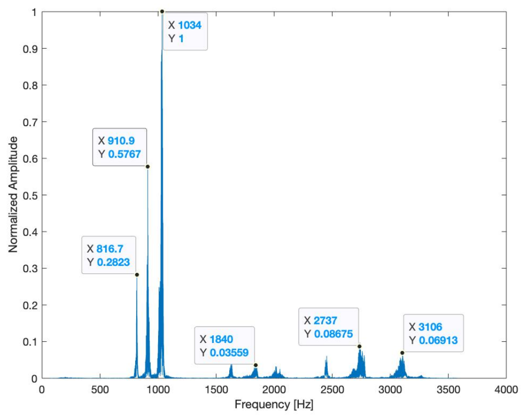

Figure 12: Fourier Transform of the Recorder Composition

Figure 12 shows that the recorder composition was played two octaves higher than the piano composition played in middle C. Also, it appears that the piece was played incorrectly as the three prominent frequencies correspond approximately to A flat (~817 Hz), B flat (~911 Hz), and C (~1034 Hz). For the piece to have been played correctly in this octave, the three frequencies should have been C (~1047 Hz), D (~1175 Hz), and E (~1319 Hz). Some overtones are present; however, unlike the piano overtones, the recorder's overtones in Figure 12 do not seem to decrease exponentially with frequency as the piano's overtones in Figure 9. The piano also has more identifiable overtones than the recorder, which contributes to the piano's 'richer' sound or *timbre* compared to the recorder. Figure 13 provides a score of the recorder composition, created using the same Shannon filter used for the piano composition; however, the recorder piece is shorter by two seconds, so the filter shifts over 140 points.

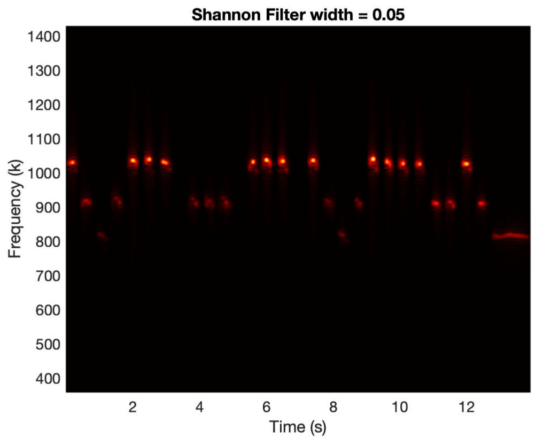

Figure 13: Score of the Recorder Composition for *Mary Had a Little Lamb*

5. Conclusions

5.1. Handel's *Messiah*

Handel's *Messiah Hallelujah Chorus* was filtered with a Gaussian, Ricker, and Shannon Gábor filter to compare the three filters. For a filter of the same width and same number of translations, both the Ricker kernel and the Shannon kernel produced spectrograms with superior time resolution than the Gaussian kernel. The Gaussian kernel should be used when computational speed is not a concern or when large frequency bandwidths are required. A Ricker wavelet has provides better time resolution than the Gaussian, and more frequency content than the Shannon filter. A Shannon filter provides the sharpest time resolution, but the smallest frequency content.

5.2. Compositions of *Mary Had a Little Lamb*

Two compositions of *Mary Had a Little Lamb*, played on the piano and recorder, were compared. The piano composition was played in the correct key (middle C) although the piano seems to be flat. The recorder composition was played two octaves higher than the piano composition, but was played with incorrect notes, likely due to a less skilled performer. The data also shows that the piano produces more overtones than the recorder and therefore has a richer timbre: producing what is often considered a more desirable tone. A Shannon filter was chosen for its superior time resolution to create a spectrogram of each rendition, which recreated the score of the piece played by each instrument.

6. References

1. "Ch. 13." *Data-Driven Modeling Et Scientific Computation: Methods for Complex Systems Et Big Data*, by J. Nathan. Kutz, Oxford Univ. Press, 2013.
2. Sound Engineering Academy. (2020). *Human Voice Frequency Range - Sound Engineering Academy*. [online] Available at: https://www.seaindia.in/human-voice-frequency-range/

APPENDIX A – Supplemental Formulae

Time-frequency spread of a Gábor transform:

$$\sigma_t^2 = \int_{-\infty}^{\infty} (\tau - t)^2 \left|g_{t,\omega}(\tau)\right|^2 d\tau$$

$$\sigma_v^2 = \int_{-\infty}^{\infty} (v - \omega)^2 \left|\tilde{g}_{t,\omega}(v)\right|^2 dv$$

- where $\sigma_t\sigma_\omega$ is independent of t and ω and is governed by the Heisenberg uncertainty principle [2]

MATLAB Functions Used

SEE APPENDIX B FOR CODE DESCRIBING THE CUSTOM FUNCTIONS USED IN THIS ANALYSIS

Full explanations and examples for each function used can be found on MathWorks' website.

- *audioread* reads in audiofiles and returns a row vector of the signal amplitude and the sampling rate of the file
- *linspace* creates a linear distributed vector of n points between the lower bound and upper bound of the distribution.
- *abs* returns the absolute value of real numbers and the real component of imaginary numbers
- *max* returns the maximum value of each column of data in a matrix and can also output the the linear index of each max value corresponding to each input matrix column
- *fftshift* shifts the Fourier transformed data to its mathematically correct location along the frequency space
- *fft* performs a Fast Fourier transforms a 1-dimensional matrix, *fft* only works on 1D vectors
- *pcolor* creates a pseudo color plot (similar to a heatmap) for an input matrix and plots it as an array of colored cells.

APPENDIX B – MATLAB CODE

hw2_main file

```
% AMATH 582 Homework 2 - Time-Frequency Analysis, Wyatt Scherer; due 2/7/20
%% Part 1 - Time-frequency Analysis of Handel's Messiah
clc; clear all; close all;
load handel
s = y'/2; % load Handel Hallelujah Chorus Sample
% y is the sampled data, Fs is the sampling rate

% plot signal sample
figure(1)
plot((1:length(s))/Fs,s);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Unfiltered Signal');

% %playback audio of signal
% p8 = audioplayer(s,Fs);
% playblocking(p8);

% Part 1 Contiuned - Build Spectograms for Handel's Messiah
% create the wave-vector domains and transformations for manipulation
n = length(s); %modern fft alg domains don't require explicit 2^n nodes
t0 = 1/Fs; ts = length(s)/Fs; tspan = linspace(t0,ts,n); t=tspan(1:n);
L = ts-t0;  k = (Fs/n)*[0:n/2 -n/2:-1]; ks = fftshift(k);

st = fft(s(1:n)); stplot = abs(fftshift(st)); smax = abs(max(st));

figure(2)
plot(ks,stplot/smax);
xlabel('Frequency [Hz]'); ylabel('Normalized Amplitude');
%title('Total');

%Build filters to sample through the frequency space
% Starting with Gabor-gaussian filters of varying width

wdm = [0.010,0.050,0.10]; %different filter widths
wdg = [0.010,0.050,0.10];
wds = [0.01,0.05,0.10];
tfilt = linspace(t0,ts,90); %generate a filter center domain changing by 0.1

%build gaussian gabor filters
ggf = build_filt(wdg,tfilt,t,'GG'); %build a gaussian filter for each width
mhf = build_filt(wdm,tfilt,t,'MHW'); %build a Mexihat filter for each width
shf = build_filt(wdm,tfilt,t,'SH'); %build a Mexihat filter for each width

%Applying the different filters to the signal data
[sggf, stggf, stggf_sft] = filter_data(s(1:n),ggf);
[sgmf, stgmf, stgmf_sft] = filter_data(s(1:n),mhf);
[sgsf, stgsf, stgsf_sft] = filter_data(s(1:n),shf);

% figure(3)
% for j =1:length(tfilt)
%     subplot(3,1,1), plot((1:length(s))/Fs,s(1:n),t,ggf(j,:,1))
%     subplot(3,1,2), plot(t,sggf(j,:,1))
%     subplot(3,1,3), plot(ks,stggf_sft(j,:,1))
%     drawnow
%     pause(0.1)
% end

%build spectogram of filtered data for width 1
figure(4)
pcolor(tfilt,ks,stggf_sft(:,:,1).'), shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Gaussian Filter width = 0.01')
xlabel('Time (s)')
ylabel('Frequency (Hz)')

%build spectogram of filtered data for width 5
figure(5)
pcolor(tfilt,ks,stggf_sft(:,:,2).'), shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Gaussian Filter width = 0.05')
xlabel('Time (s)')
ylabel('Frequency (Hz)')

%build spectogram of filtered data for width 10
figure(6)
pcolor(tfilt,ks,stggf_sft(:,:,3).'), shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Gaussian Filter width = 0.10')
xlabel('Time (s)')
ylabel('Frequency (Hz)')

%build spectogram of ricker filtered data for width 5
figure(7)
pcolor(tfilt,ks,stgmf_sft(:,:,2).'), shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Ricker Filter width = 0.05')
xlabel('Time (s)')
ylabel('Frequency (Hz)')

%build spectogram of shannon filtered data for width 5
figure(8)
pcolor(tfilt,ks,stgsf_sft(:,:,2).'), shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Shannon Filter width = 0.05')
xlabel('Time (s)')
ylabel('Frequency (Hz)')

%% Part 2 - Analysis of Piano Data
clc; clear all; close all;
%load in data
figure(9)
tr_piano = 16;   % record time in seconds
sp = audioread('music1.wav'); Fsp = length(sp)/tr_piano; sp = sp';
plot((1:length(sp))/Fsp,sp);
xlabel('Time [sec]'); ylabel('Amplitude');
```

```
title('Mary had a Little Lamb (Piano)');
drawnow
%p8 = audioplayer(sp,Fsp); playblocking(p8);

% create the wave-vector domains and transformations for manipulation
np = length(sp); %modern fft alg domains don't require explicit 2^n nodes
tp0 = 1/Fsp; tsp = length(sp)/Fsp; tspanp = linspace(tp0,tsp,np);
t=tspanp(1:np);
L = tsp-tp0; kp = (Fsp/np)*[0:np/2-1 -np/2:-1]; ksp = fftshift(kp);

stp = fft(sp); stplotp = abs(fftshift(stp)); smaxp = abs(max(stp));

figure(10)
plot(ksp([np/2+1:np/2+50000]),stplotp([np/2+1:np/2+50000])/smaxp);
xlabel('Frequency [Hz]'); ylabel('Normalized Amplitude');
%title('Frequencies of Interest, f(k)');

wdm = [0.10]; %different filter widths
wds = [0.05];
tfilt = [tp0:0.10:tsp]; %generate a filter center domain changing by 0.1

%build gaussian gabor filters
%mhf = build_filt(wdm,tfilt,t,'MHW'); %build a Mexihat filter for each width
sgf = build_filt(wdm,tfilt,t,'SH'); %build a shannon filter for each width

%Applying the different filters to the signal data
[sgmf, stgmf, stgmf_sft] = filter_data(sp,mhf);
[sgsf, stgsf, stgsf_sft] = filter_data(sp,sgf);

figure(11)
pcolor(tfilt,ksp([np/2+1:np/2+10000]),stgsf_sft(:,[np/2+1:np/2+10000],1).'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Shannon Filter width = 0.05')
xlabel('Time (s)')
ylabel('Frequency [Hz]')


%% Part 2 - Recorder data
clc; close all; clear;
figure(12)
tr_rec=14;  % record time in seconds
sr = audioread('music2.wav'); Fsr=length(sr)/tr_rec; sr = sr';
 plot((1:length(sr))/Fsr,sr);
 xlabel('Time [sec]'); ylabel('Amplitude');
 title('Mary Had a Little Lamb (Recorder)');
 %p8 = audioplayer(y,Fs); playblocking(p8);

% create the wave-vector domains and transformations for manipulation
nr = length(sr); %modern fft alg domains don't require explicit 2^n nodes
tr0 = 1/Fsr; tsr = length(sr)/Fsr; tspanr = linspace(tr0,tsr,nr);
t=tspanr(1:nr);
L = tsr-tr0; kr = (Fsr/nr)*[0:nr/2-1 -nr/2:-1]; ksr = fftshift(kr);

str = fft(sr); stplotr = abs(fftshift(str)); smaxr = abs(max(str));
```

```
figure(13)
plot(ksr([nr/2+1000:nr/2+50000]),stplotr([nr/2+1000:nr/2+50000])/smaxr);
xlabel('Frequency [Hz]'); ylabel('Normalized Amplitude');
%title('Frequencies of Interest, f(k)');

wdm = [0.10]; %different filter widths
wds = [0.05];
tfilt = [tr0:0.10:tsr]; %generate a filter center domain changing by 0.1

%build gaussian gabor filters
%mhf = build_filt(wdm,tfilt,t,'MHW'); %build a Mexihat filter for each width
shf = build_filt(wds,tfilt,t,'SH'); % build a shannon filter

%Applying the different filters to the signal data
%[sgmf, stgmf, stgmf_sft] = filter_data(sr,mhf);
[sgsf, stgsf, stgsf_sft] = filter_data(sr,shf);

%plot the spectrogram
figure(14)
pcolor(tfilt,ksr([nr/2+5000:nr/2+20000]),stgsf_sft(:,[nr/2+5000:nr/2+20000],1
).'), shading interp
set(gca,'Fontsize',[14])
colormap(hot)
title('Shannon Filter width = 0.05')
xlabel('Time (s)')
ylabel('Frequency (k)')
```

filter_data.m

```
function [sgf, stgf, stgf_sft] = filter_data(input_signal,input_filter)
%This function builds filtered data matrices based on input filters and
%returns a matrix for the filtered data, fft'd data, and the shifted fft'd
%data
%   input_signal is the signal vector to be filtered
%   input_filter, is the pre-built filter vector (see build_filt)

[jft, ift, kft] = size(input_filter);
[js, is] = size(input_signal);

% if is ~= ift
%
%     print('Wrong filter size, function will break')
%
% end

sgf = zeros(jft,ift, kft);
stgf = zeros(jft,ift,kft);
stgf_sft = zeros(jft,ift,kft);

for k = 1:kft

    for j = 1:jft

        sgf(j,:,k) = input_filter(j,:,k).*input_signal; % apply the filter to
the signal
        stgf(j,:,k) = fft(sgf(j,:,k));  % fft the time signal to get freq
        stgf_sft(j,:,k) =
abs(fftshift(stgf(j,:,k)));%/max(abs(stggf(i,:,k)));
    end

end


end
```

build_filt.m

```
function [filter] = build_filt(widths,centers,tspan,f_type)
%This function builds a 1D gabor gaussian or ricker, or shannon filter
%   Widths is a vector of filter widths (sigma)
%   Centers is a vector of the time values used for the center of the
%   filter
%   tspan is the span of the filter
%   f-type determines if a ricker filter ('MHW')or a gaussian filter ('GG')
%   is desired


% initialize filter vector
filt = zeros(length(centers),length(tspan),length(widths));

if string(f_type) == 'MHW' % Build a Ricker (mexican hat) filter

    for k=1:length(widths)

        for j = 1:length(centers)
            % build filter using mexihat function
            filt(j,:,k) = MHW_filter(widths(k),centers(j),tspan);

        end

    end



elseif string(f_type) == 'GG' % Build a Gaussian Gabor filter

    for k=1:length(widths) % go through each filter width

        for j = 1:length(centers)
            % build filter using gassian gabor function
            filt(j,:,k) = GG_filter(widths(k),centers(j),tspan);

        end

    end


elseif string(f_type) == 'SH' % Build a Shannon Gabor filter

    for k=1:length(widths) % go through each filter width

        for j = 1:length(centers)
            % build filter using gassian gabor function
            filt(j,:,k) = SH_filter(widths(k),centers(j),tspan);

        end

    end
```

```
else % only error catch is if the wrong filter type is inputted

    print('ERROR - wrong filter type')


end

filter = filt;

end
```

MHW_filter.m
```matlab
function mhf = MHW_filter(width,center,tspan)
%Creates a Gabor-Ricker filter with a specified width and center over
% time span vector tspan

A = 1; % normalizing constant - set to one for now
mhf = A.*(1-((tspan-center).^2/width.^2)).*exp(-(tspan-
center).^2/(2*width.^2)); % the function

end
```

SH_filter.m
```matlab
function filter = SH_filter(width,center,tspan)
%Creates a 1D Gabor-shannon filter with a specified width and center over
% time span vector tspan

%width is the width of the filter
%center is the vector of all centers of the filter along tspan
%tspan is the total span over which the filter applies

f = ones(size(tspan));

min = center - width; max = center + width;

z0 = find(min > tspan | tspan > max);

for i=1:length(z0)

    f(z0(i)) = 0;
end

filter = f;


end
```

GG_filter.m
```matlab
function gfilt = GG_filter(width,center,tspan)
%Creates a Gabor-Gaussian filter with a specified width and center over
% time span vector tspan
%   Detailed explanation goes here
gfilt = exp(-(1/width)*(tspan-center).^2);

end
```