Time-Frequency Analysis and Gabor Filtering of Both Simple and Complex Musical Samples

Trent Yarosevich[1]

Abstract
We use windowed time-frequency analysis, specifically windowed Fourier transforms, to examine their utility and the impact of varying features such as window-size, filter-width and structure, as well as Low-Pass frequency filtering. The objective is to make high-level comments about the kinds of information these tools let us extract from music by examining it in a frequency domain.
https://github.com/tyarosevich/AMATH_582/blob/master/HW2/yarosevich_582_hw2_final.pdf

[1] *Department of Applied Mathematics, University of Washington, Seattle*

Contents

1. Introduction and Overview

Music is comprised of oscillating signals formed by pressure waves in a medium (usually air), and thus it lends itself very naturally to analysis in a frequency domain. However, because we sense sound as the sum content of these pressure waves, the frequency content is not something we are always aware of. The interaction of these individual components is, however, a fundamental part of music (harmony).

Discussing the frequency content of music in the spatial domain requires expert knowledge of music theory. Changing to a frequency domain, however, let's us examine in detail the frequency components of a given piece of music. The problem with this is that the evolution of those frequencies in time is lost. Time-frequency analysis is an approach that allows us to get some of the best of both worlds. Thus the basic high-level claim we want to support is that we can use time-frequeny analysis to extract detailed frequency information from a piece of music while also knowing where it occurs in time. To illustrate this, the analysis will be comprised of two parts: in Part I, I will examine the famous Hallelujah Chorus from Handel's "Messiah", and use it to discuss the Gábor transform in general, as well as to study differently structured windows. In Part II, I will turn to a musically simple example, Mary Had A Little Lamb, in order to examine resolution in time and frequency, and explore the limitations if time-frequency analysis.

2. Theoretical Background

2.1 The Main Obstacle - Mathematical Uncertainty

The core of this analysis - and its obstacles - is the Fourier transform, which uses a kernel e^{-ikt} to change the basis functions of a given function $f(t)$ to sines and cosines. The transform and its inverse are as follows:

$$\begin{aligned} F(k) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-ikt} f(t)dt \\ f(t) &= \frac{1}{\sqrt{2\pi}}\int_{-infty}^{\infty} e^{ikt} F(k)dk \end{aligned} \quad (1)$$

This change of bases uses integration to describe a spatial signal, in our case sound waves, in terms of their frequency components, but in doing so it loses information about momentum or position. Intuitively, this stems from the fact that as a change in time dt gets very small, the indeterminacy of a change in frequency k will get large - we can't detect a change in pitch in an infinitely small amount of time, which is why the Fourier transform integrates across the entire domain. Similarly, if determinacy of frequency is perfect, we would be describing a perfect sine or cosine function with an infinite domain in time[1]. From a purely mathematical point of view, we can obvserve that in the Fourier transform itself, all the dependence on time is integrated 'out' of the data.

2.2 The Solution - Windowed Transforms

The solution to this conundrum is to used 'windowed' transforms. Informally, this means that we can take the transform of a section of music localized in a certain span of time, thus giving a time-frame for its occurence. One way of doing this is to use the Gábor Transform G, which is an intuitve extention of the Fourier transform that introduces a new kernel localizing information around a specific time τ[2]. Note that the function $g(\tau - t)$, which is used to localize a particular time, is asumed to be real and symmetric, with a 2-norm of 1:

$$g_{\tau,\omega}(\tau) = e^{i\omega\tau} g(\tau - t) \text{ the kernel}$$
$$G[f](t,\omega) = \bar{f}(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau} d\tau \qquad (2)$$

One way to envision this is that the integration across all τ is 'sliding' the window across the signal, and extracting the frequency information inside each window. This then gives us the possibility of describing frequency content with respect to time, also known as a spectrogram. Given this resulting data, the original signal can then be recovered using the inverse transform. While we will not make use of this here, since the objective is to analyze data in the frequency domain, it is included for completness as well as to make the instructive observation that the inverse transform is a double integral, unlike the inverse Fourier transform, since it must integrate across all frequency and time shifting components:

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

2.3 Limitations

The fundamental problem of mathematical uncertainty is not avoided by the Gábor transform. If the 'window' that the transform slides across the signal is very narrow, we will recover frequency components at very specific timeframes, i.e. we will have good resolution in time. However, it should be very clear that this neglects lower frequency components whose period is significantly longer than the window. For example the window in Figure 1 would capture nearly all the frequencies present across about two seconds - very good resolution in frequency, but very poor in time (a lot happens in two seconds of music). Similarly, the opposite holds and a very wide window collects more frequency components, but is less localized in time. Illustrating this limitation and working around it will be one of he principle focuses of the analaysis.

[1]"Feature Column from the AMS." American Mathematical Society, www.ams.org/publicoutreach/feature-column/fcarc-uncertainty.

[2]Kutz, Jose Nathan. Data-Driven Modeling and Scientific Computation. Methods for Complex Systems and Big Data. Oxford University Press, 2013.

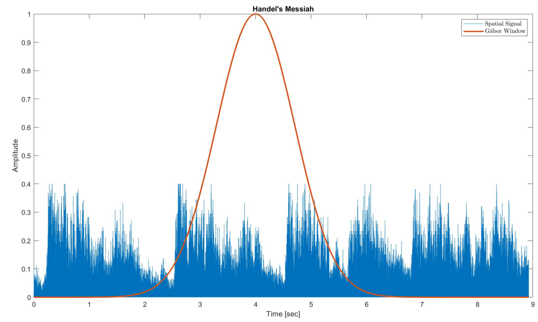


Figure 1. An illustration of a relatively 'wide' transform window

3. Algorithm Implementation and Development

In each section of this analysis, the same general algorithmic approach will be used several times, and so I will discuss its formulation once. In each case, data will be imported and processed to produce a spectrogram, with variations in parameters to conduct the analysis, but with the same overall structure each time.

3.1 Discretized Gábor Filtering

At the outset, we must note that because we are analyzing digital sound information, we must use a discretized Gábor transform, producing a lattice of time and frequency as shown in Figure 2[3]. This lattice occurs in uniform steps in both time

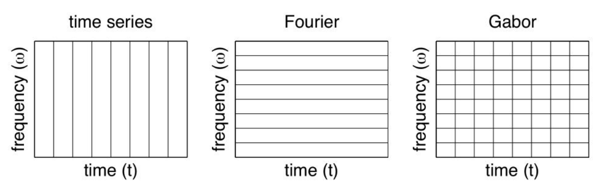


Figure 2. Motivation for discretized Fourier and Gábor Transforms

and frequency, which can be scale arbitrarily, but is indexed in m,n. The following discretized form of the transform is then used:

$$\bar{f}(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt \qquad (4)$$

3.2 Sampling rate and Time Domain

When the data is imported, it is an arbitrary vector of information, and in order to construct sensible domains in time and frequency, we have to know the sampling rate. In part 1, this rate is included with Matlab's built-in Handel file, but in Part II it is easily determined by dividing the length of the data vector by the duration, in seconds, of the sample. A time mesh can then be constructed by making an a vector with

[3]Ibid.

a number of points arbitrarily equal to the length of the data vector and dividing it by the sampling rate. This then gives suitably scaled time-mesh.

3.3 Frequency Domain

3.3.1 Fast Fourier Transform

In order to quickly process discretized data, we will be using the Fast Fourier Transform (FFT), which assumes a 2π periodic domain. Because this is a periodic domain, the last element of the spatial domain is ommitted when constructing the K domain mesh, since this last element would be a repetition of the zero wave number.

3.3.2 Frequency Units

In Part I, I will simply scale the frequency domain mesh to a 2π periodic domain, so that the resulting domain is given in wave numbers k. This is accomplished by creating a mesh in time, and then noting the flipped domain of the FFT, which is then scaled by $\frac{2\pi}{L}$, as shown in the code-listing in **Appendix II** lines 26-7.
In Part II, however, we will be using a (Hz) based frequency domain since we will be considering musical pitch. In this case, we will then retain the scaling from the spatial domain and simply divide the frequency domain by the length L of the sample, to put the frequency in units of cycles per second.

3.4 Discretized Gábor Transform

The implementation of the discrete Gábor transform is relatively straightforward. In essence, we are using iteration to 'slide' a window across the data, as opposed to integrating in the continuous case.

3.4.1 Step 1: slide vector and spectrogram matrix

First, a vector is declared to determined how many windowed transforms are taken, and how far apart they are. Over and undersampling in this regard will be considered in the analysis, but in general, a larger number of smaller steps produces more accurate resolution in time. Second it is important to declare a matrix of zeros to hold the resulting data. The matrix rows correspond to the amplitude values across a frequency domain, and the columns to time. The number of rows is equal to the length of the tslide vector, and the columns to the length of the original time vector used to construct the frequency domain.

3.4.2 Step 2: The window

Next, the equation governing the window, $\bar{g}_{m,n}(t)$, must be chosen. There are many possible choices, and in the analysis we will use a Gaussian window, a so-called 'Mexican Hat' filter, and a Shannon filter. The construction of the Gaussian and Mexican Hat filters is fairly straightforward - their equations evaluated at a shifted time domain vector.

The Shannon filter is a little more complicated, but still fairly simple. To create a filter that steps up and then down after an arbitrary width, we essentially slide the t-mesh itself by the current value in the tslide vector, then apply this vector to a boolean evaluation of $t < \frac{w}{2}$ and $t > -\frac{w}{2}$. This returns a boolean vector with zeros anywhere except a step centered around w which can then be multiplied by the data just as the other filters are. These filters are shown in Figure 3.

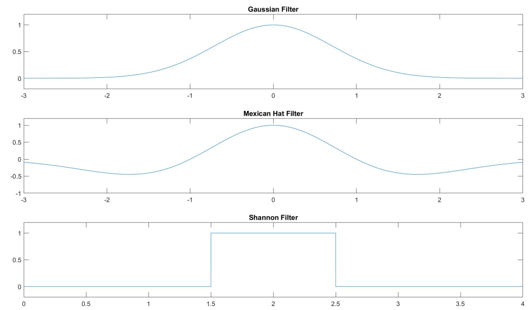


Figure 3. Plots of the filters used in the analysis

3.4.3 Step 3- Iteration

The actual iteration across the windows is a done with a 'for' loop indexed from 1 to the length of the tslide vector. In each iteration we then evaluated the g function at the time mesh minus the current tslide value, multiply it componentwise with the data, and then take the FFT of the result and store it as a row of the holder matrix.

3.4.4 Step 4 - The Spectrogram

The resulting data is then used to create a spectrogram to visualize the frequency components over time. To make this more efficient, we can note that because we are transforming real spatial data, the negative frequencies will simply mirror the positive and can be discarded. In the case of Part II, we can take this a step further and observe that the frequency content of the samples falls in a small range. Thus by making observations from a plot in the frequency domain of the entire recording, we can ignore empty frequency domains by creating an index vector between two particular values. This is done by passing the shifted frequency mesh *ks* as an argument to the *find()* function in Matlab with boolean conditions for the upper and lower limit. The resulting vector can then be passed as an index when plotting the data. The data is then plotted using the *pcolor()* function in Matlab, which we use to plot the frequencies on the y-axis, time on the x-axis, and then color density using the matrix of amplitude data from the Gábor filtering.

3.4.5 Step 5 - Overtone filtering

The last algorithmic issue to discuss is the filtering of timbre. The timbre of an instrument is comprised of many elements, including overtones (integer multiples of the fundamental frequency of the note) as well as the structure of the fundamental frequency's peak, i.e. it's 'attack' and 'decay' as well as the small frequency components immediately around it. In the analysis of Part II, this timbre will be filtered out to examine the fundamental frequencies more closely.

To do this I simply identified the peak fundamental frequencies using a spatial plot of the sample, then centered three

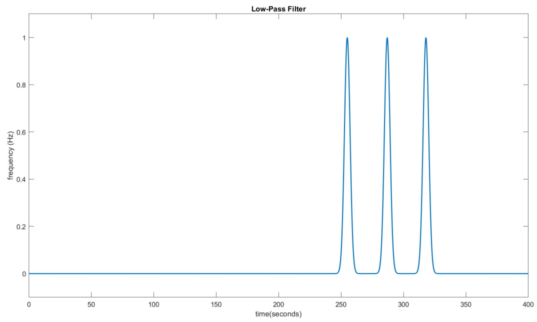
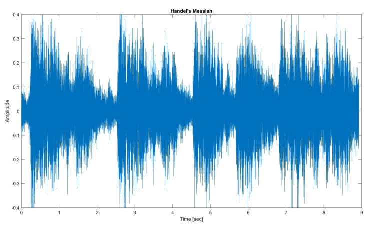
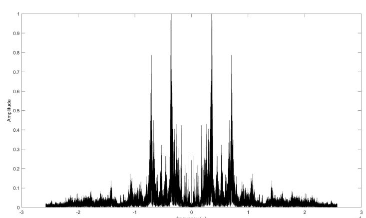


Figure 4. The three low-pass filters for removing overtones



(a) Spatial Structure



(b) Frequency Structure

Figure 5

Gaussian filters around these peaks. These filters were then evaluated across the shifted frequency domain, added together as shown in Figure 4 , and multiplied by the transformed and shifted data. The data was then shifted back to the FFT domain, inverse transformed, and Gábor filtered as described above. Note that the width of the Gaussian filters is controlled with parameter w, and a wide filter can be used to remove only the overtones, or a very narrow one to also filter out the attack and decay of the fundamental frequencies.

4. Computational Results

4.1 Handel's "Messiah"

To begin analyzing Handel's "Messiah" it is useful to motivate the need for frequency analysis by looking Figure 5 and simply observe what was noted at the outset: the spatial plot tells us little about the frequency content, and the frequency-domain plot tells us nothing about frequency components over time.

To begin a time-frequency analysis, we first consider two spectrograms with opposite goals - one uses a very narrow window in order to achieve excellent resolution in time, and the other a wide window to achieve excellent resolution in space. In both cases, the window slides in increments of 0.1 seconds, and Figure 6 shows the dramatic difference in approach. In the frequency-resolved spectrogram we see the multitude of frequencies, clustered around 10 or more that are likely fundamental frequencies, and a large number of overtones. The time-resolved spectrogram on the other hand retains evidence of the cadence of the piece, with gaps between each "Hallelujah" evident in the graph.

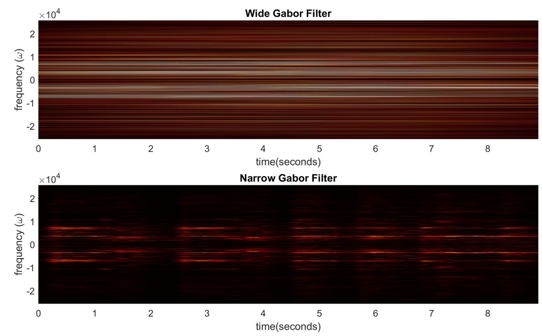


Figure 6. A narrow versus wide filter

The next objective was to compare the result of the different types of filters. The three are compared in Figure 7 with good resolution in time. The resulting spectrograms suggest that the Shannon filter is particularly good at achieving resolution in time while retaining more frequency information than the other filters, with an extreme example of a Shannon filter with a width of .1 - the same as the slide width - shown in Figure 8. In general, the different filters performed fairly similarly, however at moderately small window sizes, the Shannon filter again seems to give slightly better resolution in time. This can be see with careful examination in figure 7, in which the the Shannon filter manages to capture the slight amplitude reduction between the first and second syllables of "Hallelujah".

I also investigated the effects of over and undersampling, which had the sensible result that large steps between the transform windows resulted in very poor resolution in time, giving the worst of both worlds - poor resolution in both time and frequency, as shown in Figure 9.

4.2 Part II - Mary Had a Little Lamb

This section narrowed in on simpler samples in order to analyze the frequency components themselves in a more qualitatively meaningful way than the "Messiah", which is harmonically complex. First, we considered the frequency domains of the two instruments, Piano and Recorder. Figure 10 shows clear differences. In particular, the piano has notable overtones, whereas the recorder has a slow attack, probably caused by the exhalation pressure of the player building to a peak. Figure 11 shows the various spectrograms produced with narrow and wide Gábor filters both before and after timbre filtering. As expected, the narrow windows give us good resolution in time, and we can see the individual notes

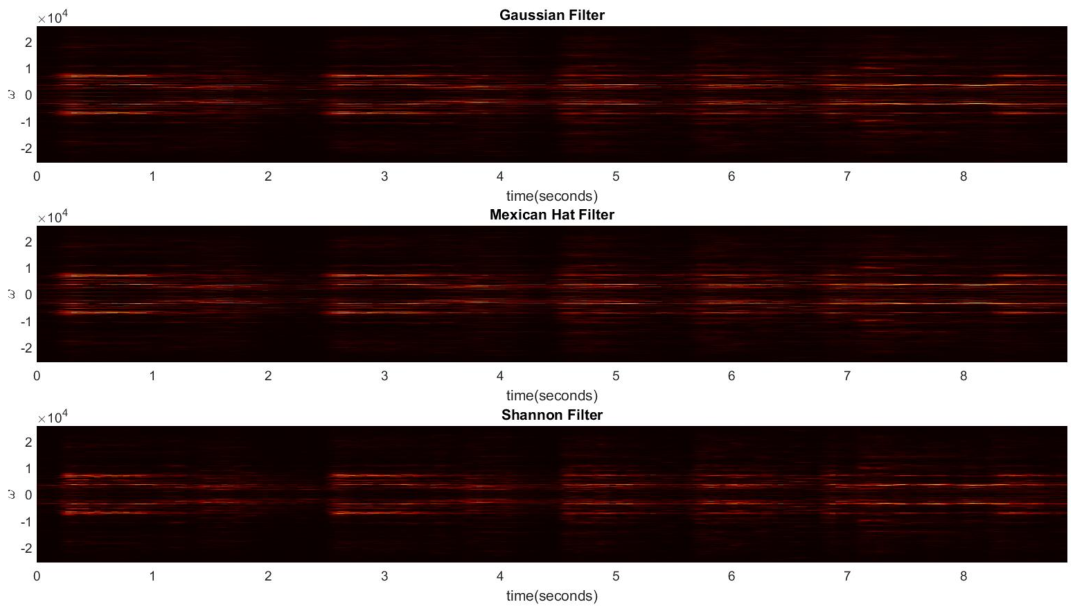


Figure 7. Comparison of different Gábor Filters g

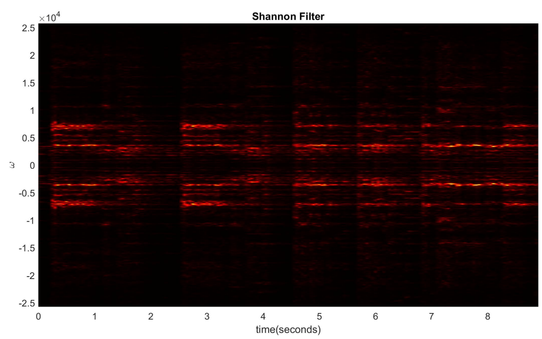


Figure 8. A finely time-resolved shannon filter

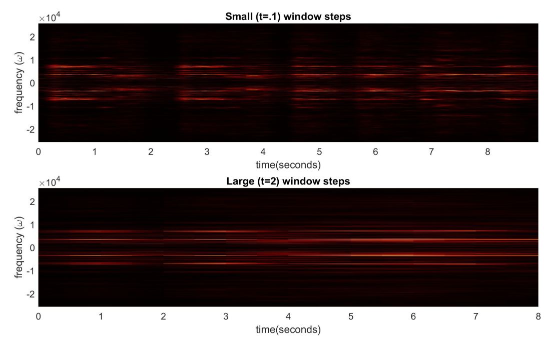


Figure 9. Under/over-sampling

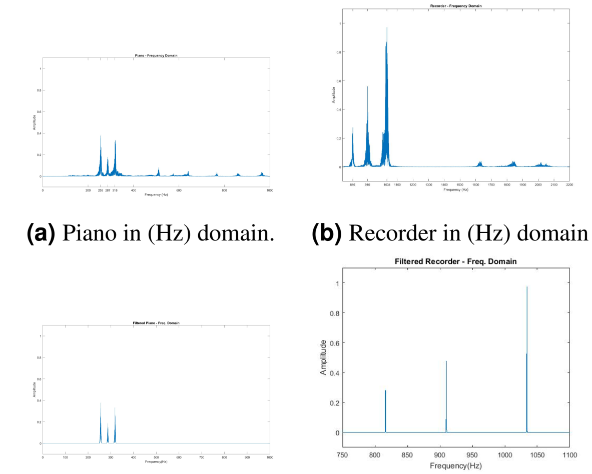
of the melody. Even with a wide filter, however, it is a little difficult to see the fundamental frequencies due to the timbre. After timbre filtering, we then see exceptional resolution in frequency, particularly for the recorder.

5. Summary and Conclusions

The high-level goals of the analysis were satisfied, particularly in Part II. We saw that Gábor filtering was effective at isolating frequency information and charting its development over time. Perhaps the most important demonstration of this is found in Figure 6 in which we have attained resolution in time and frequency side-by-side, such that one could examine frequency content at a given time, and use the good time resolution spectrogram to associate it with specific portions of music, e.g. one could hone in on the frequency content of the ”Ha” in ”Hallelujah” if one were so inclined.

For example, even without the ubiquitous knowledge of the melody and cadence of ”Mary Had A Little Lamb”, we can deduce from Figure 11(e) and (g) that the melody is playing notes approximately three times every two seconds, at frequencies of about 318 Hz, 287 Hz, and 256 (Hz) - namely, E4, D4, and C4 on a slightly out of tune (flat) Piano. Similarly, we can deduce that the Recorder is playing notes around 1035 Hz, 912 Hz, and around 820 Hz - which are somewhat close to C5 flat, B4 flat, and A4 flat.

In addition to the overall goals, I encountered a few issues that merit mention. For one, I found that using filters to take the original sample close to a pure tone was sometimes problematic, as removing too much of the signal could make any resolution in time difficult. This also wound up somewhat circular in its logic, since if we know what the fundamental frequency is, we could simply generate a pure tone anyway. Reducing a complex piece to pure tones might be helpful, but it would require some clever implementation to distinguish fundamentals from overtones.

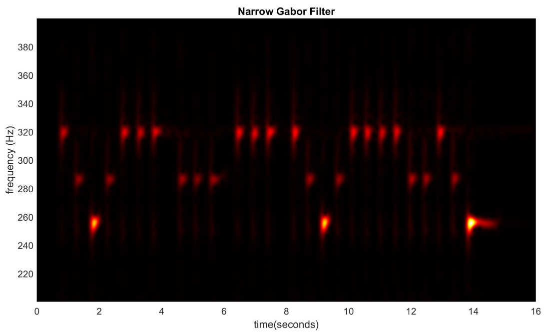


(a) Piano in (Hz) domain. **(b)** Recorder in (Hz) domain

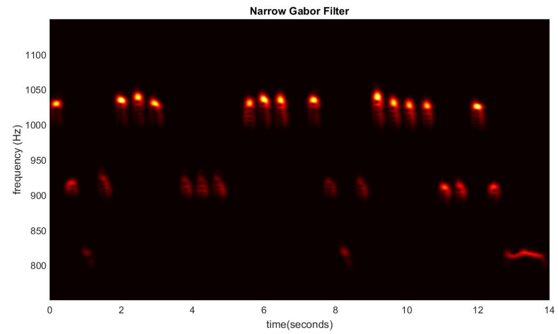
(c) Piano after Timbre Filtering **(d)** Recorder after Timbre Filtering

Figure 10. Before and after Timbre Filtering

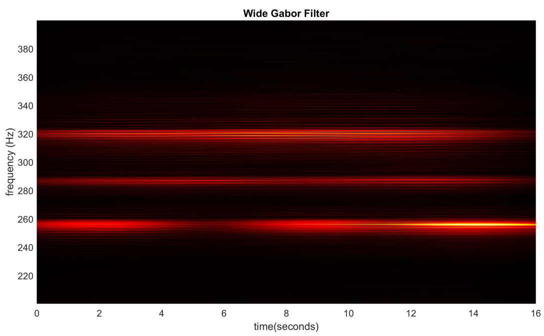
Another interesting development is found in the subtle timbre in the piano sample, which can be seen in 11(a). I am unsure what these frequencies represent, but they appear to occur around previously played notes and occur at precisely the same time as the currently played notes. I can only speculate that perhaps these are frequencies caused when the previously played strings in the piano are still vibrating slightly, and some sympathetic vibration from the currently played note kicks it up into barely detectable amplitude.



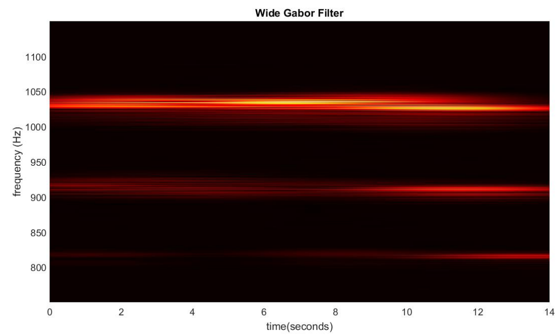
(a) Piano with narrow window



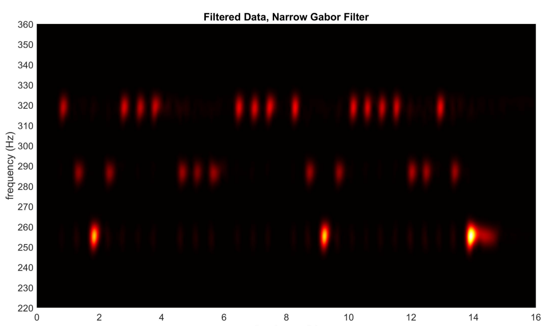
(b) Recorder with narrow window



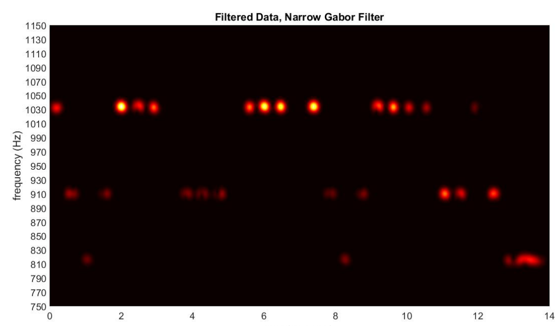
(c) Piano with wide window



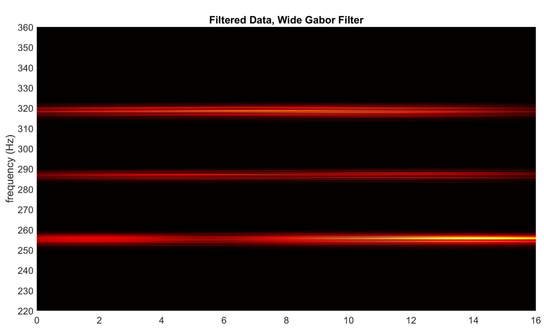
(d) Recorder with wide window



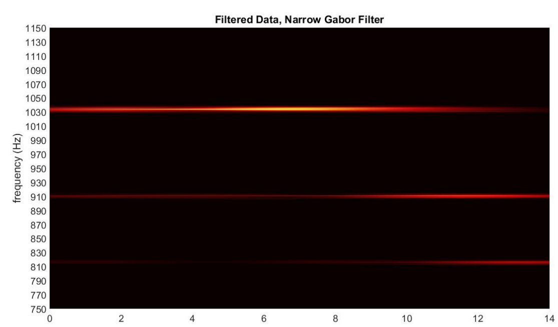
(e) Filtered piano, narrow window



(f) Filtered recorder, narrow window



(g) Filtered piano, wide window



(h) Filtered recorder, wide window

Figure 11. bla

Appendix 1

linspace(a,b,n): returns a vector in the span $[a,b]$ with n points.

fftshift(data)/ifftshift(data): Shifts data between zero-centered wave number domain and the domain used by the Fast Fourier Transform.

abs(data): Returns the absolute value of the passed object.

pcolor(a, b, c): Pcolor accepts two vectors which it plots on the x and y axes, and then accepts matrix of size a x b which it uses to convey a third dimension through color density in accordance with a variety of optional color maps.

Appendix 2

```
clc; clear all; close all;

% Load and plot the The Messiah segment

load handel
v = y'/2;
plot((1:length(v))/Fs,v);
xlabel("Time [sec]");
ylabel("Amplitude");
title("Handel's Messiah");
axis([0 9 0 1])
%% Plot with a 'window'
t_mesh = (1:length(v))/Fs;
hold on
plot(t_mesh, exp(-(t_mesh - 4).^2), 'LineWidth', 2)
legend('Spatial Signal', "G\'abor Window", 'Interpreter', 'latex')

%% Playback

p8 = audioplayer(v,Fs);
playblocking(p8);

%% Build the time and frequency domains

L = length(v)/Fs; n = length(v);
t2 = linspace(0, L, n+1); t = t2(1:n);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);
v = v(1:end - 1);
v_t = fft(v);

%% Plot in freq. domain of original file
close all
plot(ks,abs(fftshift(v_t))/max(abs(v_t)),'k'); % axis([-50 50 0 1])
%set(gca,'Fontsize',[14])
xlabel('frequency (\omega)'), ylabel('Amplitude')

%% Gabor Window Iteration/Slide
close all
%figure(4)
tslide=0:0.1:L;

messiah_spec_1 = zeros(length(tslide), length(t)-1);
```

```
messiah_spec_2 = zeros(length(tslide), length(t)-1);

% Two different bands to compare
w1 = .1;
w2 = 100;

for j=1:length(tslide)
    g=exp(-w1*(t(1:end-1)-tslide(j)).^2); % Gabor
    v_g= g.*v;
    v_g_t=fft(v_g);
    messiah_spec_1(j, :) = abs(fftshift(v_g_t));
end
for j=1:length(tslide)
    g=exp(-w2*(t(1:end-1)-tslide(j)).^2); % Gabor
    v_g= g.*v;
    v_g_t=fft(v_g);
    messiah_spec_2(j,:) = abs(fftshift(v_g_t));
end

figure(5)
subplot(2,1,1)
pcolor(tslide,ks,messiah_spec_1.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
xlabel('time(seconds)'), ylabel('frequency (\omega)')
title('Wide Gabor Filter');

subplot(2,1,2)
pcolor(tslide,ks,messiah_spec_2.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
xlabel('time(seconds)'), ylabel('frequency (\omega)')
title('Narrow Gabor Filter')

%% Under versus oversampling
close all
%figure(4)
tslide1=0:0.1:L;
tslide2=0:1:L;
messiah_spec_1 = zeros(length(tslide1), length(t)-1);
messiah_spec_2 = zeros(length(tslide2), length(t)-1);


w = 100;

for j=1:length(tslide1)
    g=exp(-w*(t(1:end-1)-tslide1(j)).^2); % Gabor
    v_g= g.*v;
    v_g_t=fft(v_g);
    messiah_spec_1(j, :) = abs(fftshift(v_g_t));
end
for j=1:length(tslide2)
    g=exp(-w*(t(1:end-1)-tslide2(j)).^2); % Gabor
    v_g= g.*v;
    v_g_t=fft(v_g);
    messiah_spec_2(j,:) = abs(fftshift(v_g_t));
end

subplot(2,1,1)
pcolor(tslide1,ks,messiah_spec_1.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
```

```
xlabel('time(seconds)'), ylabel('frequency (\omega)')
title('Small (t=.1) window steps');

subplot(2,1,2)
pcolor(tslide2,ks,messiah_spec_2.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
xlabel('time(seconds)'), ylabel('frequency (\omega)')
title('Large (t=2) window steps')

%% Plot the different filters
close all

x_mesh = linspace(-3, 3, 100);
y_g = exp(-20*(x_mesh(1:end)).^2);
y_m = (1 - x_mesh.^2).*exp(-10*(x_mesh).^2);
t_shift = t(1:end -1)- 2;
y_sh = t_shift < 1/4 & t_shift > -1/4;

subplot(3,1,1)
plot(x_mesh, y_g)
axis([-3 3 -.2 1.2])
title("Gaussian Filter")

subplot(3,1,2)
plot(x_mesh, y_m)
axis([-3 3 -1 1.2])
title("Mexican Hat Filter")


subplot(3,1,3)
plot(t(1:end-1), y_sh)
axis([0 4 -.2 1.2])
title("Shannon Filter")


%% Plot Spectorgrams with Multiple types of filters
close all

tslide=0:0.05:L;
w = 100;

% Spatial resolution with the shannon filter is pretty outstanding
% with this filter width - the doubled 'hallelujah's are pretty clear
% in the spectrogram.
w_sh = .1;
messiah_sp_gauss = zeros(length(tslide), length(t)-1);;
messiah_sp_mex = zeros(length(tslide), length(t)-1);;
messiah_sp_shan = zeros(length(tslide), length(t)-1);;

for j=1:length(tslide)
    g=exp(-w*(t(1:end-1)-tslide(j)).^2);
    v_g= g.*v;
    v_g_t=fft(v_g);
    messiah_sp_gauss(j,:)=abs(fftshift(v_g_t));
end

for j=1:length(tslide)
    g=(1 - (t(1:end -1) - tslide(j)).^2).*exp(-w*(t(1:end-1)-tslide(j)).^2);
    v_g= g.*v;
```

```
    v_g_t=fft(v_g);
    messiah_sp_mex(j,:)=abs(fftshift(v_g_t));
end

for j=1:length(tslide)

    % Shift the t-mesh by the slide value, then
        use this
    % to return a vector of booleans depending on
        whether or not
    % the value in the t-mesh is greater than the
        -width/2 AND less
    % than the width/2. This centers a square
        filter at tslide(j) or tau.
    t_shift = t(1:end -1) - tslide(j);
    g = t_shift < w_sh/2 & t_shift > -w_sh/2;
    v_g= g.*v;
    v_g_t=fft(v_g);
    messiah_sp_shan(j,:)=abs(fftshift(v_g_t));
end

%% Plot the data using pcolor
close all

subplot(3,1,1)
pcolor(tslide,ks,messiah_sp_gauss.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
xlabel('time(seconds)'), ylabel('\omega')
title('Gaussian Filter');

subplot(3, 1, 2)
pcolor(tslide,ks,messiah_sp_mex.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
xlabel('time(seconds)'), ylabel('\omega')
title('Mexican Hat Filter')

subplot(3,1,3)
pcolor(tslide,ks,messiah_sp_shan.'),
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
xlabel('time(seconds)'), ylabel('\omega')
title('Shannon Filter')

%% Part II

clc; clear all; close all;

% tr_piano=16; % record time in seconds
% y_p=audioread('music1.wav'); Fs=length(y_p)/
    tr_piano;
% plot((1:length(y_p))/Fs,y_p);
% xlabel('Time [sec]'); ylabel('Amplitude');
% title('Mary had a little lamb (piano)'); drawnow
% p8 = audioplayer(y_p,Fs); playblocking(p8);
% figure(2)
tr_rec=14; % record time in seconds
y_p=audioread('music2.wav'); Fs=length(y_p)/tr_rec
    ;
plot((1:length(y_p))/Fs,y_p);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
% p8 = audioplayer(y_p,Fs); playblocking(p8);

%%
L = 14; n = length(y_p);
```

```
t2 = linspace(0, L, n+1); t = t2(1:n);
 % k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);
k = (1/L)*[0:(n/2-1) -n/2:-1];
ks = fftshift(k);

%% Plot the Piano in Frequency Domain
y_p_t = fft(y_p);
plot(ks, -abs(fftshift(y_p_t))/max(y_p_t), 'LineWidth', 1)
axis([750 2200 -.1 1.1])
xticks([700 816 910 1034 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000 2100 2200])
xticklabels({'700', '816', '910', '1034', '1100', '1200', '1300', '1400',  '1500', '1600', '1700', '1800', '1900', '2000', '2100', '2200'})
xlabel('Frequency (Hz)'), ylabel('Amplitude')
title('Recorder - Frequency Domain');
%% Check width of gabor filter
w = .1;
plot(t,exp(-w*(t(1:end)-8).^2))

%% Gabor Window Iteration/Slide
close all; clc;
%figure(4)
tslide=0:0.05:L;
piano_spec = zeros(length(tslide), length(t));
w = 100;
y_p = y_p';


for j=1:length(tslide)
    g=exp(-w*(t(1:end)-tslide(j)).^2); % Gabor
    y_g= g.*y_p;
    y_g_t=fft(y_g);
    piano_spec(j,:)= abs(fftshift(y_g_t));
end

% Range of frequency to plot, based on observation of the frequency domain
% plot.
idx = find(ks > 750 & ks < 1150);

pcolor(tslide,ks(idx),piano_spec(:,idx).')
shading interp
set(gca,'Fontsize',[14])
colormap(hot)
%axis([0 16 -.2 .2])
xlabel('time(seconds)'), ylabel('frequency (Hz)')
title('Narrow Gabor Filter');

%%
y_p_t_s = fftshift(y_p_t);
% [B, I] = maxk(y_p_t_s, 3);
% This wound up being annoying because the top three max freq were
% all around the first (loudest) note.

% Gaussian filters to remove all overtones. The k1/2/3 are
% simply found by examining a frequency domain plot of the sample.

% Piano Peaks
% k1 = 255;
% k2 = 287;
% k3 = 318;

% Recorder Peaks
k1 = 816;
```

```
k2 = 910;
k3 = 1034;

w = .1;

f1 = exp( -w *(ks - k1).^2);
f2 = exp( -w *(ks - k2).^2);
f3 = exp( -w *(ks - k3).^2);

y_p_filtered = (y_p_t_s.* (f1 + f2 + f3)');
%% Plot the filters To confirm width
% Note the filter width matters, since a very narrow filter
% will drastically reduce the amplitude and produce less contrast
% in the pcolor plot with sound versus silence.
plot(ks, f1 + f2 + f3)
axis([750 1200 -.1 1.1])
xlabel('time(seconds)'), ylabel('frequency (Hz)')
title('Low-Pass Filter');
%% Plot the filtered frequency domain

close all
plot(ks, -abs(y_p_filtered)/max(y_p_filtered))
axis([750 1100 -.1 1.1])
xlabel('Frequency(Hz)'), ylabel('Amplitude')
title('Filtered Recorder - Freq. Domain');
%% Now a spectrogram of the filtered data
close all; clc;
%figure(4)
tslide=0:0.05:L;
filtered_spec = zeros(length(tslide), length(t));


% Two different bands to compare
w = .1;
w_sh = .1;

y_p_f = ifft(ifftshift(y_p_filtered));


for j=1:length(tslide)
    g=exp(-w*(t(1:end)-tslide(j)).^2); % Gabor

    % Shannon Filter (gaussian seems to work better)
    % t_shift = t(1:end) - tslide(j);
    % g = t_shift < w_sh/2 & t_shift > -w_sh/2;

    y_g= g'.*y_p_f;
    y_g_t=fft(y_g);
    filtered_spec(j,:) = abs(fftshift(y_g_t));
end

%% Plot using pcolor

pcolor(tslide,ks(idx),filtered_spec(:,idx).')
shading interp
set(gca,'Fontsize',[14], 'YLim', [750 1150], 'YTick', [750:20:1150])
% set(gca,'Fontsize',[14])
colormap(hot)
%axis([0 16 -.2 .2])
xlabel('time(seconds)'), ylabel('frequency (Hz)')
title('Filtered Data, Narrow Gabor Filter');

% Results are perfect. The Piano is out of tune, ranging from around 5
% to 10 Hz flat.
```

```
%% Play the filtered data

% Note the playback sounds like pure tones, just
    as we would expect
p8 = audioplayer(y_p_f,Fs); playblocking(p8);
```