Time-Frequency Analysis of One-Dimensional Audio Data

Maxwell Weil

Abstract

In this homework, we will be examining techniques for representing one-dimensional data in both the time and frequency domains simultaneously. We will first overview of the concepts and strategies used in this investigation, providing a theoretical background for this paper. Next, we will explore our data in both the time and frequency domains using MATLAB software coupled with signal and wavelet analyzer toolboxes. Ultimately, a better understanding and appreciation for the computational methods to visualize and analyze time-frequency data will be established.

I. Introduction

The analysis of signals in the time and frequency domains is inherently problematic. To view a signal in the time domain, we lose the ability to see frequencies. Similarly, in the frequency domain, we can't distinguish when certain information is occurring. This issue limits the ability to fully analyze and breakdown data. To combat this problem, a mathematical conversion called the Gabor transform was created. This technique computes a Fourier transform over small periods of time, allowing preservation of both time and frequency information. However, there are limits to the effectiveness of the Gabor transform.

In this work, the Gabor transform will be used on several datasets and analyzed in its entirety. Additionally, related methods of time-frequency analysis will be demonstrated and compared to this transform. Using these strategies, we will process one-dimensional sound data taken from two different instruments. In doing this, we aim to visualize how this data is changing in both the time and frequency domains. The differences and similarities between the sound samples will be compared, granting a comprehensive understanding of the data. Before these analyses, a we will numerically explore the algorithms and methods that we will use.

II. Theoretical Background

The Gabor transform is the centerpiece of this homework, so it is necessary to explain its fundamentals. As mentioned above, the Gabor transform computes the Fourier transform of a signal over various periods of time. Mathematically, the Gabor transform is shown in equation 1 below. Note that all equations have been adapted from [1].

$$G_x(t,\omega) = \int_{-\infty}^{\infty} x(\tau)g(\tau - t)e^{-i\omega\tau}d\tau \qquad \text{EQ. 1}$$

Looking at this formula, there is a clear resemblance to the Fourier transform equation. However, the most notable differences are the variables ω and τ substituting for k and t, respectively, and the inclusion of a new function, g(τ-t). ω is essentially the same as k, although the units have changed from 1/seconds to radians/seconds. τ is now the time, while t represents a shift. This is better understood in the context of g(τ-t), which is a window centered at time t (shifted along τ). In the Gabor transform, g(τ-t) take the shape of a Gaussian window, as outlined in equation 2.

$$g(\tau - t) = e^{-a*(\tau - t)^2} \qquad \text{EQ. 2}$$

In this equation, a dictates the width of the window, and is inversely proportionally to the window size. This window is then multiplied by the signal of interest in the time domain, x(τ), and then transformed using a Fourier transform. The window shifts along the time signal, generating many Fourier transforms that make up what is called a spectrogram. Essentially, a spectrogram is just a representation of many Fourier transforms over different time windows. The Gabor transform that creates the spectrogram is also known as the short-time Fourier transform (STFT) for this reason. As with the Fourier transform, the STFT is invertible, with the inverse shown in equation 3.

$$x(\tau) = \frac{1}{2\pi}\frac{1}{||g||^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G_x(t,\omega)g(\tau - t)e^{i\omega\tau}d\omega dt \qquad \text{EQ. 3}$$

The inverse transform is again similar to the inverse of the Fourier transform. Note that a double integral is used instead, as integration is over all frequencies and all time. However, in order to computationally utilize the STFT, we also have to have a discretized form, which can be seen in equation 4.

$$G_x(m,n) = \int_{-\infty}^{\infty} x(t)g(t - nt_0)e^{i2\pi m\omega_0 t}dt \qquad m = \frac{\omega_0}{\omega} \qquad n = \frac{t_0}{\tau} \qquad \text{EQ. 4}$$

In this form, both frequency and time have been discretized into m and n, respectively. This results in a basic change of variables throughout the equation. This equation will form the basis of our work in this assignment. Additionally, we will be using different types of windows within the STFT. In addition to the typical Gaussian window, a Shannon window and Mexican hat window will be analyzed. The equations for the Shannon and Mexican hat windows are also given in equations 5 and 6, respectively.

$$s(\tau - t) = \mathrm{H}\left(\tau - t + \frac{w}{2}\right) - H(\tau - t - \frac{w}{2}) \qquad \text{EQ. 5}$$

$$m(\tau - t) = (1 - a(\tau - t)^2)e^{\frac{-a*(\tau-t)^2}{2}} \qquad \text{EQ. 6}$$

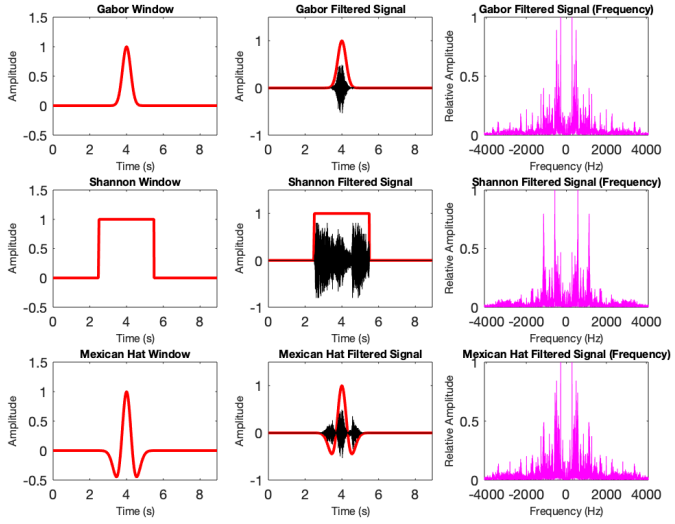

Figure 1. Plots of various window type for use in the Gabor transform. The first column shows each window individual, the seconds shows the window multiplied by the investigated audio signal, "Handel," and the third shows the resulting frequency domain of the signal. Each row looks at a different window style, including the traditional Gabor (gaussian), Shannon and Mexican Hat windows. Note that the Gabor transform essentially functions as many Fourier transforms of a filtered time signal, with the filter moving along in time steps.

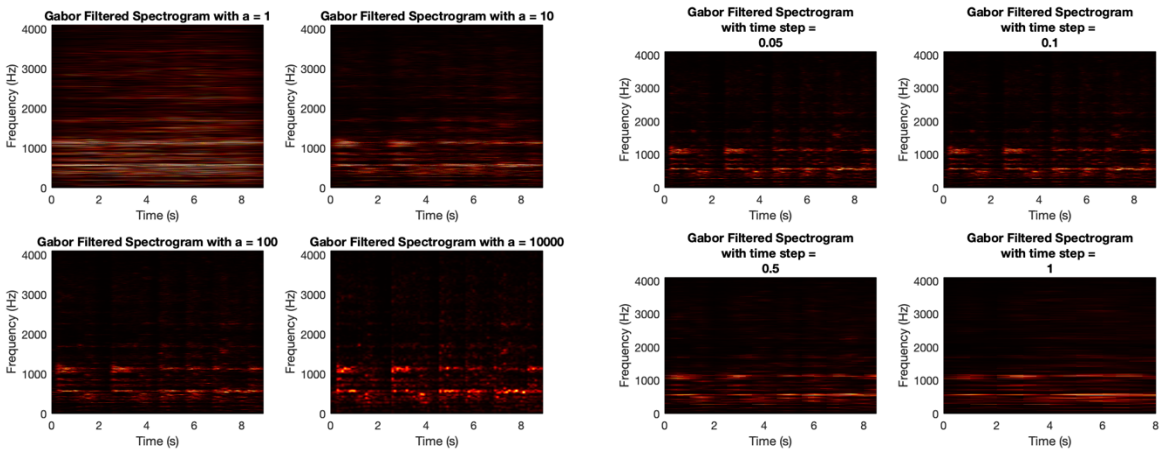

Figure 2. Spectrograms produced by using various window sizes and time steps with the Gabor transform are shown. The traditional Gabor (gaussian) window is used, the equation for which is shown in equation 2. In the left most four plots a time step of 0.1 seconds was used, while a values varied from 1 to 10000. In the the right most four plots, an a value of 100 was kept constant while the time step size changed from 0.05 seconds to 1 second.

The Shannon window, s(τ-t), makes use of the Heaviside function, H, where negative values are set to 0, and positive values are set to 1. This window ends up appearing as a square pulse of width w. The Mexican hat window is similar to the Gabor window, though it aims to capture additional low frequencies by including additional oscillations. Examples of these three windows and their impact on filtering in time are shown in Figure 1.

Using these different windows, we will be aiming to computing various Gabor transforms on a few one-dimensional audio signals. However, the Gabor transform has its limitations. Namely, due to the uncertainty relationship between the time and the frequency domain, the resolution of this transform is limited. Filtering more precisely in time will give us finer time resolution, but we will lose the ability to see longer waveforms representing low frequencies. Likewise, in order to capture low frequencies, the time window must be large, removing our ability to discern precise time points. When the window is expanded infinitely, the Gabor transform actually reverts to the Fourier transform, due to losing all time resolution. Related to this issue, is the size of time steps. Small time steps result in oversampling, where data becomes choppy and challenging to relate to adjacent points. Very large time steps blur together frequencies, making it harder to tell when different frequencies are happening in time. Because of these issues, appropriate window sizes and time steps must be selected before performing time-frequency analysis. Figure 2 explores these concepts visually, by comparing the spectrograms of Gabor transforms with various windows widths and time steps.

III. Algorithm Implementation and Development

In this paper, several aspects of the Gabor transform are explored through the use of MATLAB software. First, windows used were visualized in the time domain, as shown in Figure 1 (lines 25-30). These windows were multiplied by an audio signal of interest, Handel, and illustrated in both time and frequency domains (lines 37-103). Note that the frequencies used throughout this work were scaled by a factor of 1/(sampling rate). This is because we wished to view the frequencies in Hz, rather than in radians per second as done previously.

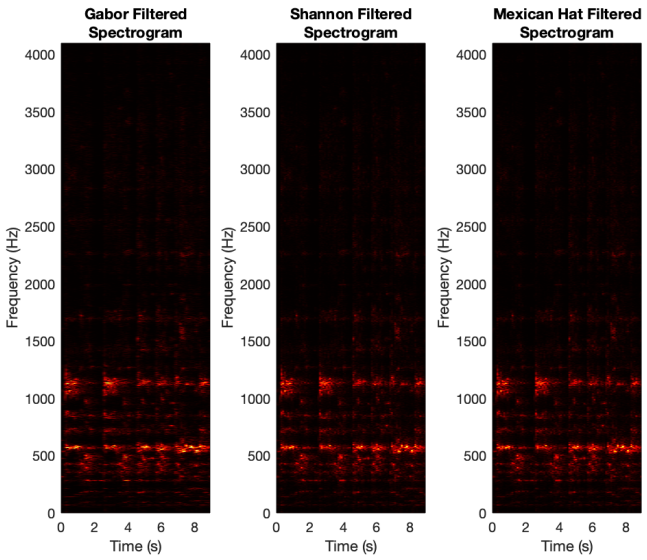

Figure 3. Resulting spectrograms for Gabor transforms computed with various window types. Time steps for the transform were all 0.1 seconds, while window widths varied. Both the traditional Gabor window and the Mexican hat window used an a = 100, while the Shannon window used a window width of 0.1 seconds.

To compute the transform, and audio signal Handel was stepped through in time, multiplied by a window centered about that time point, and then transformed into the frequency domain (lines 123-129). The resulting frequency information was stored in a matrix, with rows representing the frequency data and columns containing the time point (line 132). This matrix was then visualized as a spectrogram, using the pcolor command in MATLAB (lines 136-143). This process will be used several times throughout this investigation.

As shown in Figure 2, components of the Gabor transform were inspected, and it was found that a gaussian window with a = 100 and a transform with a time step of 0.1 seconds were most optimal for our purposes. Following this, the gaussian, Shannon, and Mexican hat windows were compared, again using the Handel audio signal. The results for this can be seen in Figure 3. The same process as before was followed, though for different window equations.

After these initial analyses, two additional audio samples were analyzed using optimized methods. Each audio signal was a recording of "Mary Had a Little Lamb," though played on different instruments (a piano and a recorder). These signals were read into MATLAB and downsampled to avoid computational difficulties (lines 266-270). Then, they were transformed into spectrograms using the Gabor transformation with a gaussian window (lines 296-300). Additionally, the maximal frequency at each time step was filtered (lines 303-311) using a gaussian window with a = 0.01. This was done to avoid overtones interfering with the reproduction of music scores for each instrument (this will be explained further in the results). The raw spectrograms without overtone filter can be seen in Figure 4 (lines 318-325).

Spectrograms with filtered overtones were generated and additional steps were taken to extract the music score for both the piano and recorder audio signals from these. To do this, maximal frequencies were processed by finding values over a certain threshold (relating to a note being played), separated by gaps of relative silence (releasing of a note, lines 329-335). These tones were then transposed into alphabetical notes, and the resulting music score was recreated, as shown in Figure 5.

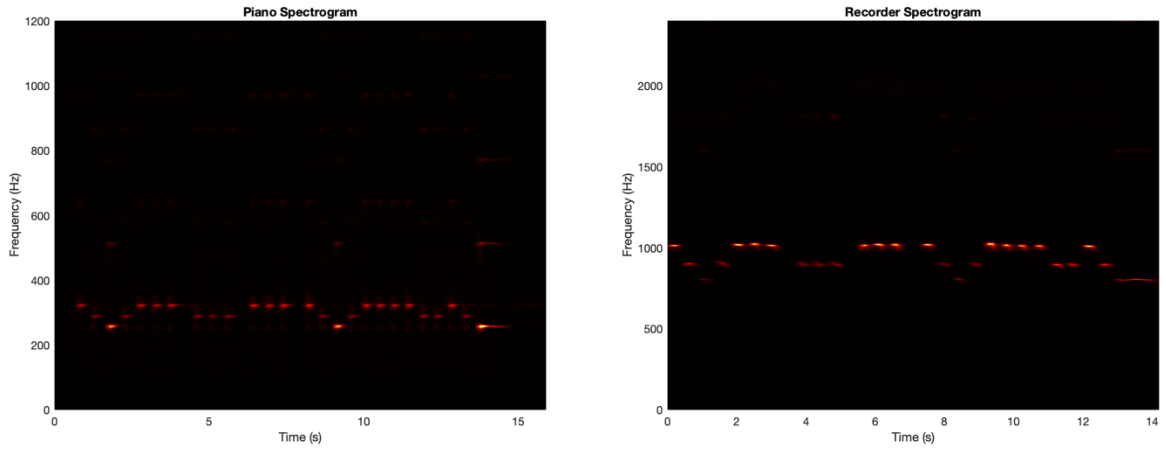

Figure 4. Raw spectrograms of a piano and recorder playing "Mary Had a Little Lamb." A traditional Gabor filter was used with an a value of 100. Note the differences between each recording. While the recorder clearly plays at a higher frequency, different overtones can also be seen. If a note is played at a frequency ω, then overtones will occur at integer multiples of ω (2ω, 3ω, etc.).

IV. Computational Results

From our investigation, a number of interesting results were found. As detailed previously, our initial experiments aimed to examine various properties of the Gabor transform on the audio sample, Handel. Different time steps, window sizes, and window styles were explored. Overall it was shown that a moderately sized time step and window size were most desirable. This prevented undersampling and oversampling, and also avoided having poor resolution in either the time or frequency domain.

The comparison of window styles was not as straightforward, however. All windows gave fairly similar spectrogram information. This may be due to the sample data being used. In this instance, no window gave a clear advantage over another because the results were primarily exploratory. In a more rigorous analysis of some type of data, one window may be preferred due to different properties. For example, the Mexican hat window may be able to more readily detect frequencies that resemble its shape than the Shannon window.

Following these results, new audio recordings were used in an attempt to reconstruct a music score for "Mary Had a Little Lamb." As mentioned in Figure 4, one notable difference between these recordings are the presence of different overtones. Overtones are frequencies that occur at integer multiples of a center frequency. For example, a middle A tone at 440Hz may have overtones at 880Hz and 1320Hz. In different instruments, different overtones are present at varying amplitudes. This property is known as the timbre of an instrument, and it's why a piano and recorder playing the same note at the same frequency sound very different. This property is also impossible to view in either the time or frequency domain, meaning spectrograms are necessary to realize this difference between the recordings. When the overtones are removed, the remaining pure tone will always sound the same. In our work, we filtered out any overtones before recreating the music scores for both instruments, in order to clean up the audio data.

The reproduced music scores show that the piano and recorder were playing the same pattern of notes, but at different frequencies (this is also apparent from the spectrograms). The notes are labelled alphabetically in Figure 5, along with the frequencies they were found at. As before, frequencies were only scaled by a factor of 1/(sampling rate) to ensure that they could be translated into Hz tones.

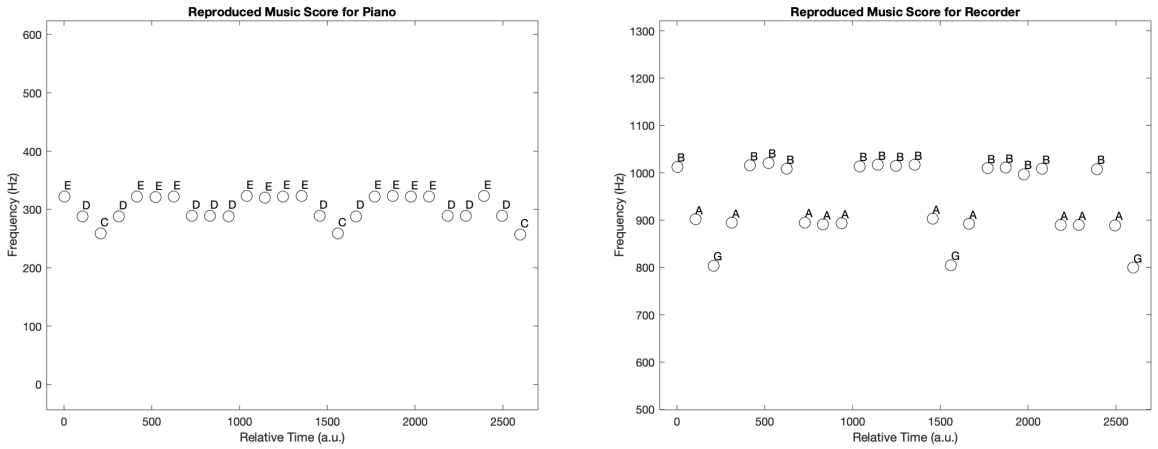

Figure 5. Reproduced music scores for each instrument. Note that the time axis is arbitrarily scaled and is only used to denote the sequence of notes occurring. Both the notes and frequencies for each instrument are different, though both follow the same pattern throughout the song. While the recorder score has a greater difference between each frequency, the aligned notes are each a whole step apart, as in the piano score. Alphabetical notes have been estimated based on the approximate frequencies found.

V. Summary and Conclusions

From the techniques used in this assignment, we have been able to explore the mechanisms of the Gabor transform, interpret spectrograms, and practically apply these principles to analyze sample data. The use of windowing/filtering, Gabor transforms, and spectrograms highlights how these techniques can be applied to a multitude of problems. Time-frequency analysis as a whole allows for the access of information that previously could not be seen. Overall, this assignment served to show how this powerful tool works, and how it can be adapted to analyze data in various scenarios.

References

[1] J. Nathan Kutz, *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data.* Oxford University Press, 2013.

Appendix A. MATLAB Functions Used

`heaviside(x)` – Unit step function, outputs 0 when x < 0, 1 when x > 0, and 1/2 when x = 0.
`pcolor(x,y,c)` – 2D plot of x and y values, where c corresponds to color values of each cell.
`shading interp` – Colors a cell by interpolating a color value across the cell.
`colormap map` – Applies a predefined gradient of color values to the current figure
`audioread(filename)` – Reads in an audio file and outputs a data and sampling rate
`downsample(x,n)` – Decreasing the sampling rate of x by factor of n.
`diff(x)` – Finds the difference between adjacent elements of x.
`strings(n)` – Create an empty string array of size n.
`text(x,y,str)` – Adds chosen string to x and y data points on a plot.

Appendix B. MATLAB Code

```
% Maxwell Weil
% AMATH 482
% HW2
clear; close all; clc;

% Loading sample music and sampling rate, reorienting vector
load handel
music = y';

% Setting sample number, duration of music, time vector,
% And unshifted/shifted frequency vectors
samples = length(music);
duration = samples/Fs;
time = [(1:samples)/Fs];
freqs = (1/duration)*[0:(samples-1)/2 -(samples-1)/2:-1];
shifted_freqs=fftshift(freqs);


% Defining constant for plotting static windows
a_ex = 10;
width_ex = 3;
sigma_ex = 10;
tau = 4;

% Defining equations for gabor, shannon, and mexican hat windows
gabor_window=exp(-a_ex*(time-tau).^2);
shannon_window = heaviside(time-(tau-width_ex/2))...
    -heaviside(time-(tau+width_ex/2));
mexican_hat_window = (1-sigma_ex*(time-tau).^2)...
    .*exp(-1*sigma_ex*(time-tau).^2/2);

% Creating plots of static windows and filtered signal in
% Frequency and time domains
figure

% Gabor window subplots
subplot(3,3,1)
plot(time, gabor_window, 'r', 'LineWidth', 2)
axis([0 duration -0.5 1.5])
title('Gabor Window', 'FontSize', 6)
xlabel('Time (s)', 'FontSize', 6)
ylabel('Amplitude', 'FontSize', 6)
subplot(3,3,2)
plot(time, gabor_window, 'r', 'LineWidth', 2)
hold on
plot(time, gabor_window.*music, 'k')
axis([0 duration -1 1.5])
title('Gabor Filtered Signal', 'FontSize', 6)
xlabel('Time (s)', 'FontSize', 6)
ylabel('Amplitude', 'FontSize', 6)
subplot(3,3,3)
plot(shifted_freqs,abs(fftshift(fft(gabor_window.*music)))...
    /max(abs(fft(gabor_window.*music))),'m');
axis([-Fs/2 Fs/2 0 1])
title('Gabor Filtered Signal (Frequency)', 'FontSize', 6)
xlabel('Frequency (Hz)', 'FontSize', 6)
ylabel('Relative Amplitude', 'FontSize', 6)

% Shannon window subplots
subplot(3,3,4)
```

```
61    plot(time, shannon_window, 'r', 'LineWidth', 2)
62    axis([0 duration -0.5 1.5])
63    title('Shannon Window', 'FontSize', 6)
64    xlabel('Time (s)', 'FontSize', 6)
65    ylabel('Amplitude', 'FontSize', 6)
66    subplot(3,3,5)
67    plot(time, shannon_window, 'r', 'LineWidth', 2)
68    hold on
69    plot(time, shannon_window.*music, 'k')
70    axis([0 duration -1 1.5])
71    title('Shannon Filtered Signal', 'FontSize', 6)
72    xlabel('Time (s)', 'FontSize', 6)
73    ylabel('Amplitude', 'FontSize', 6)
74    subplot(3,3,6)
75    plot(shifted_freqs,abs(fftshift(fft(shannon_window.*music)))...
76        /max(abs(fft(shannon_window.*music))),'m');
77    axis([-Fs/2 Fs/2 0 1])
78    title('Shannon Filtered Signal (Frequency)', 'FontSize', 6)
79    xlabel('Frequency (Hz)', 'FontSize', 6)
80    ylabel('Relative Amplitude', 'FontSize', 6)
81
82    % Mexican hat window subplots
83    subplot(3,3,7)
84    plot(time, mexican_hat_window, 'r', 'LineWidth', 2)
85    axis([0 duration -0.5 1.5])
86    title('Mexican Hat Window', 'FontSize', 6)
87    xlabel('Time (s)', 'FontSize', 6)
88    ylabel('Amplitude', 'FontSize', 6)
89    subplot(3,3,8)
90    plot(time, mexican_hat_window, 'r', 'LineWidth', 2)
91    hold on
92    plot(time, mexican_hat_window.*music, 'k')
93    axis([0 duration -1 1.5])
94    title('Mexican Hat Filtered Signal', 'FontSize', 6)
95    xlabel('Time (s)', 'FontSize', 6)
96    ylabel('Amplitude', 'FontSize', 6)
97    subplot(3,3,9)
98    plot(shifted_freqs,abs(fftshift(fft(mexican_hat_window.*music)))...
99        /max(abs(fft(mexican_hat_window.*music))),'m');
100   axis([-Fs/2 Fs/2 0 1])
101   title('Mexican Hat Filtered Signal (Frequency)', 'FontSize', 6)
102   xlabel('Frequency (Hz)', 'FontSize', 6)
103   ylabel('Relative Amplitude', 'FontSize', 6)
104   %% Effect of Different Window Sizes on Spectrogram
105
106   % Defining window size constants for sliding gabor window
107   a_list = [1, 10, 100, 10000];
108
109   % Creating time steps to move window along
110   tslide=0:0.01:duration;
111
112   % Initializing spectrogram matrix
113   gab_spec_music = zeros(length(tslide),samples);
114
115   % Looping through different sized windows for gabor-filtered spectrogram
116   figure
117   for i = 1:length(a_list)
118
119       % Looping through time step vector
120       for j=1:length(tslide)
121
```

```
122           % Creating gabor window at time step
123           gabor_window=exp(-a_list(i)*(time-tslide(j)).^2);
124
125           % Filtering signal in time
126           gab_filt_music=gabor_window.*music;
127
128           % Transforming signal to frequency
129           freq_gab_filt_music=fft(gab_filt_music);
130
131           % Adding frequency data to spectrogram at each time step
132           gab_spec_music(j,:) = fftshift(abs(freq_gab_filt_music));
133
134       end
135           % Plotting spectrograms for each sized window
136           subplot(2,2,i)
137           pcolor(tslide,shifted_freqs,gab_spec_music.')
138           ylim([0 max(freqs)])
139           title(['Gabor Filtered Spectrogram with a = ', num2str(a_list(i))])
140           ylabel('Frequency (Hz)')
141           xlabel('Time (s)')
142           shading interp
143           colormap hot
144   end
145   %% Effect of Step Size on Spectrogram
146
147   % Defining window size constant
148   a = 100;
149
150   % Creating varying time steps to move window along
151   step_list = [0.05, 0.1, 0.5, 1];
152
153   % Looping through different sized time steps for gabor-filtered spectrogram
154   figure
155   for i = 1:length(step_list)
156
157       % Setting new time step size
158       tslide=0:step_list(i):duration;
159
160       % Initializing spectrogram matrix
161       gab_spec_music = zeros(length(tslide),samples);
162
163       % Looping through time step vector
164       for j=1:length(tslide)
165
166           % Creating gabor window at time step
167           gabor_window=exp(-a*(time-tslide(j)).^2);
168
169           % Filtering signal in time
170           gab_filt_music=gabor_window.*music;
171
172           % Transforming signal to frequency
173           freq_gab_filt_music=fft(gab_filt_music);
174
175           % Adding frequency data to spectrogram at each time step
176           gab_spec_music(j,:) = fftshift(abs(freq_gab_filt_music));
177
178       end
179           % Plotting spectrograms for each sized window
180           subplot(2,2,i)
181           pcolor(tslide,shifted_freqs,gab_spec_music.')
182           ylim([0 max(freqs)])
```

```
183           title({'Gabor Filtered Spectrogram'; 'with time step = '; ...
184                 num2str(step_list(i)))
185           ylabel('Frequency (Hz)')
186           xlabel('Time (s)')
187           shading interp
188           colormap hot
189   end
190   %% Effect of Different Window Types on Spectrogram
191
192   % Defining optimized constants for shannon and mexican hat windows
193   a = 100;
194   width = 0.1;
195   sigma = 100;
196
197
198   % Initializing spectrogram matrices for each window type
199   gab_spec_music = zeros(length(tslide),samples);
200   shan_spec_music = zeros(length(tslide),samples);
201   mex_spec_music = zeros(length(tslide),samples);
202
203   % Looping through time steps, with optimized
204   % Window sizes, for spectrogram comparison
205   figure
206   for j=1:length(tslide)
207
208      % Creating each window at specified time step
209      gabor_window=exp(-a*(time-tslide(j)).^2);
210      shannon_window = heaviside(time-(tslide(j)-width/2))-heaviside(time-
(tslide(j)+width/2));
211      mexican_hat_window = (1-sigma*(time-tslide(j)).^2).*exp(-1*sigma*(time-
tslide(j)).^2/2);
212
213      % Filtering and transforming signal with gabor window
214      gab_filt_music=gabor_window.*music;
215      freq_gab_filt_music=fft(gab_filt_music);
216
217      % Filtering and transforming signal with shannon window
218      shan_filt_music=shannon_window.*music;
219      freq_shan_filt_music=fft(shan_filt_music);
220
221      % Filtering and transforming signal with mexican hat window
222      mex_filt_music=mexican_hat_window.*music;
223      freq_mex_filt_music=fft(mex_filt_music);
224
225      % Adding frequency data to each spectrogram at each time step
226      gab_spec_music(j,:) = fftshift(abs(freq_gab_filt_music));
227      shan_spec_music(j,:) = fftshift(abs(freq_shan_filt_music));
228      mex_spec_music(j,:) = fftshift(abs(freq_shan_filt_music));
229   end
230
231   % Plotting spectrograms for each window type
232   % Gabor spectrogram
233   subplot(1,3,1)
234   pcolor(tslide,shifted_freqs,gab_spec_music.')
235   ylim([0 max(freqs)])
236   title({'Gabor Filtered'; 'Spectrogram'})
237   ylabel('Frequency (Hz)')
238   xlabel('Time (s)')
239   shading interp
240   colormap hot
241
```

```
% Shannon spectrogram
subplot(1,3,2)
pcolor(tslide,shifted_freqs,shan_spec_music.')
ylim([0 max(freqs)])
title({'Shannon Filtered'; 'Spectrogram'})
ylabel('Frequency (Hz)')
xlabel('Time (s)')
shading interp
colormap hot

% Mexican hat spectrogram
subplot(1,3,3)
pcolor(tslide,shifted_freqs,mex_spec_music.')
ylim([0 max(freqs)])
title({'Mexican Hat Filtered'; 'Spectrogram'})
ylabel('Frequency (Hz)')
xlabel('Time (s)')
shading interp
colormap hot

%% PROBLEM 2, Analyzing A Music Score
clear; close all; clc;

% Reading in audio data and sampling frequency
[y,Fs] = audioread('music2.wav');

% Downsampling audio for fast computing, note that Fs is also adjusted
y = downsample(y,4);
Fs = Fs/4;

% Reorienting audio vector, setting sample number, duration of music,
% Time vector, and unshifted/shifted frequency vectors
song = y';
samples = length(song);
duration = samples/Fs;
time = [(1:samples)/Fs];
freqs = (1/duration)*[0:samples/2-1 -samples/2:-1];
shifted_freqs=fftshift(freqs);

% Setting selected filter constants and time step vector
a = 100;
a2 = 0.01;
tslide=0:0.1:duration;

% Initializing spectrogram, maximum amplitude, and tone
% At maximum amplitude matrices
spec_song = zeros(length(tslide),samples);
max_amplitude = zeros(length(tslide),1);
tone_at_max = zeros(length(tslide),1);

% Looping through time steps
for j=1:length(tslide)

    % Creating window and filtering audio
    gabor_filter=exp(-a*(time-tslide(j)).^2);
    filt_song=gabor_filter.*song;

    % Transforming to frequency domain
    filt_freq_song=fft(filt_song);

    % Finding the maximum amplitude, and the frequency at which it occurs
```

```
303       [high, idx] = max(abs(filt_freq_song));
304       tone_at_max(j) = freqs(idx);
305       max_amplitude(j) = high;
306
307       % Creating filter in frequency domain around center frequency
308       overtone_filter = exp(-a2*(freqs-freqs(idx)).^2);
309
310       % Filtering out overtones around center frequency
311       pure_freq_song = overtone_filter.*filt_freq_song;
312
313       % Adding frequency data to each spectrogram at each time step
314       spec_song(j,:) = fftshift(abs(filt_freq_song));
315   end
316
317   % Plotting spectrogram
318   figure
319   pcolor(tslide,shifted_freqs,spec_song.')
320   ylim([0 2400])
321   title('Recorder Spectrogram')
322   ylabel('Frequency (Hz)')
323   xlabel('Time (s)')
324   shading interp
325   colormap hot
326
327   % Finding where the maximum amplitudes are over a specified threshold,
328   % In order to determine true notes are being played
329   threshold_logic = max_amplitude>mean(max_amplitude)/2;
330
331   % Removing repeated values, by finding difference between adjacent elements
332   high_low_shift = diff(threshold_logic) < 0;
333
334   % Locating corresponding frequencies for each note
335   notes_freq = tone_at_max(high_low_shift);
336
337   % Initializing text string for notes
338   notes_text = strings(length(notes_freq),1);
339
340   % Looping through frequncies and determining alphabetical note,
341   % This process has been shortened to only search for a few specific notes
342   for i = 1:length(notes_freq)
343       if notes_freq(i) < 1100 && notes_freq(i) > 980
344           notes_text(i) = 'B';
345       elseif notes_freq(i) < 980 && notes_freq(i) > 850
346           notes_text(i) = 'A';
347       elseif notes_freq(i) < 850 && notes_freq(i) > 750
348           notes_text(i) = 'G';
349       elseif notes_freq(i) < 330 && notes_freq(i) > 300
350           notes_text(i) = 'E';
351       elseif notes_freq(i) < 300 && notes_freq(i) > 280
352           notes_text(i) = 'D';
353       else
354           notes_text(i) = 'C';
355       end
356   end
357
358   % Creating vector to plot notes
359   note_scaling = linspace(1, 100*length(notes_freq), length(notes_freq));
360
361   % Plotting frequencies and corresponding alphabetical note, keep in mind
362   % That the x-axis is arbitrary, and notes are not actually equally spaced
363   figure
```

```
364   plot(note_scaling,notes_freq, 'ko', 'MarkerSize', 10)
365   text(note_scaling,notes_freq+20, notes_text)
366   axis([(min(note_scaling)-100) (max(note_scaling)+100)...
367       (min(notes_freq)-300) (max(notes_freq)+300)])
368   title('Reproduced Music Score for Piano')
369   xlabel('Relative Time (a.u.)')
370   ylabel('Frequency (Hz)')
```