Audio Signal Time-Frequency Analysis Using Gábor Transforms

Seong Hyun Han

February 7, 2020

Abstract

In this assignment, Gábor transformation is used to conduct time-frequency analysis on various audio files. The time and frequency information were gained using a filter window at a specific point in time, allowing to obtain the related frequency information. The window parameters along with different window filter types were explored. This transform was then applied to create the music scores for a certain music audio signal after filtering out the overtones created from the specific instruments used to play the music.

Sec. I Introduction and Overview

The main motivation for this assignment is to explore and analyze the time-frequency signatures of three short audio signals: *Handel's Messiah*, *Marry had a little lamb* on the piano, and *Marry had a little lamb* on the recorder. *Handel's Messiah* was loaded directly from MATLAB and the two *Marry had a little lamb* audio signals, provided as **.wav** files, were generated using an iPhone. For the first part of this assignment, the frequency signatures in *Handel's Messiah* at certain moments in time will be found by creating a filter and translating it across the length of the audio signal. The filter, or window, captures the various frequencies in the signal at a given moment in time. This will allow us to generate a plot of various frequencies with respect to time, where a color gradient defines the amplitude of the frequencies present. This type of plot is known as a spectrogram. To explore the changes in the plot's time and frequency resolutions, varying window widths, overlap between windows, and different types of windows will be explored. For the second part of this assignment, music scores for both *Marry had a little lamb* audio signals will be reproduced by plotting the frequencies in each signal with respect to time. The timber of each instrument will also be compared and later filtered to extract its base notes.

Sec. II. Theoretical Background

In order to approach this assignment, several key mathematical concepts must be understood. The first concept is the Gábor Transform, also known as the *short-time Fourier Transform* (STFT). The cardinal concept of the Gábor Transform is that it decomposes a signal in the time domain into different time frames. For each time window, the Fourier transform of the signal is taken to extract the frequencies present during that specific time frame. The other time frames will be attenuated to zero. This allows for the extraction of both the frequency and time information of the signal, which contrasts the Fourier Transform that only extracts frequency information. Viewing a signal as a function, $f(\tau)$, the Gábor Transform can be defined as the integration from $\tau \in [-\infty,\ \infty]$ of a function multiplied by a time filtering window $\bar{g}(\tau - t)$ and $e^{\pm ikx}$, which relates to Euler's formula. However, by making some general assumptions of g being real and symmetric with $||g(t)|| = 1$ and $||g(\tau - t)|| = 1$ ($|| \cdot ||$ is the L_2 norm), the Gábor Transform is can be simplified (Eqn. 1). The time filtering window centers the Fourier integral around $t = \tau$, where the parameter τ slides the filter window across the entire length of the signal [1].

$$G[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-i\omega\tau} d\tau = (f, g_{t,w}) \qquad \text{(Eqn. 1)}$$

Since the Gábor Transform uses discretized time and frequency domains in practice, the discrete version of the transform is given by Eqn. 2 and Eqn. 3 where $\nu = m\omega_o$ and $\tau = nt_o$.

$$g_{m,n}(t) = e^{-i2\pi m\omega_0 t} g(t - nt_o) \qquad \text{(Eqn. 2)}$$

$$\tilde{f}_g(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt = (f, g_{m,n}) \qquad \text{(Eqn. 3)}$$

Another concept to note about the Gábor Transform is that the Heisenberg Uncertainty Principal applies due to its fixed time filtering window. An increase in frequency resolution (increase in filter window) decreases the time resolution and *vice versa*. In order to ameliorate this tradeoff, different filter window types, such as wavelets, can be used. One such wavelet is the Mexican hat wavelet (Eqn. 4) [2] which has relatively great frequency and time resolution due to its small time-bandwidth (fast decay rate controlled by σ) with similarity to the Gaussian function (Eqn. 5) [1]. Additionally, a step-function filter (Eqn. 6) called the Shannon filter (w = width of the filter) will be used to explore the resolution tradeoffs.

$$\psi(t) = c\left(1 - \left(\frac{t}{\sigma}\right)^2\right)e^{-\frac{t^2}{2\sigma^2}} \qquad \text{(Eqn. 4)}$$

$$F(k) = e^{-\tau(k-k_o)^2} \qquad \text{(Eqn. 5)}$$

$$\mathrm{H(t)} = \begin{cases} 0, & \mathrm{n} < 0 \\ 1, & \mathrm{n} \geq 0 \end{cases} \qquad \text{(Eqn. 6)}$$

$$\mathrm{S(t) = H(t + w/2) - H(t - w/2)}$$

Sec. III. Algorithm Implementation and Development – Part I

This problem is solved in MATLAB by using the methods mentioned above. Starting with **Part 1.1** in **Appendix B. MATLAB codes**, a portion of *Handel's Messiah* is loaded directly from MATLAB. The audio signal is transposed and stored as a variable *v*. The signal was sampled with a sampling frequency of *Fs* and a total number of samples of *length(v)*. The computational time domain was thus defined as *L* = *length(v)*/*Fs* which equals the total time duration of the music. The number of Fourier modes *n* equal the total number of samples. The time domain is defined from 0 to L with *n* + 1 intervals. This time interval is then converted into a frequency interval with n Fourier modes. This frequency interval must be scaled by a factor of $\frac{1}{L}$, in order to have frequency in terms of hertz and not the usual 2π periodic angular frequency (ω). The frequency components are adjusted to account for n being odd. Afterwards, they are shifted using *fftshift()* to have the value zero at the center and stored in the variable *ks*.

In order to create a Gábor Transform spectrogram, the width parameter *a1* was set to 50, and translation parameter τ was set to 0.1 by making it the interval of *tslide*, where *tslide(j)* functions as τ in a for loop. The Gábor filter/window was made using the Gaussian function (Eqn. 5). For each time frame in the for loop, the window is multiplied by the audio signal *v*, transformed into the frequency domain using *fft()* and stored in matrix *vgt_spec_g*. The filtered frequency signals are plotted as a spectrogram using *pcolor()*, where *ks* is the y-axis and the time intervals are the x-axis. For **Part 1.2**, in order to explore the effect of the window width on the time and frequency resolutions, various width parameters were used (*a* = 100, 10, and 1) for the Gábor filter/window and their respective spectrograms plotted. For **Part 1.3**, in order to explore the effect of undersampling versus oversampling, the transition parameter τ was changed in the Gábor filter/window to 1.5 and 0.1, respectively. For **Part 1.4**, in order to explore the various window filter types, the Mexican hat wavelet filter was made according to (Eqn. 4) with t - *tau_slide* to translate the window, $\sigma = 0.1$, and $c = 1$ (all of the filters have a maximum of one for comparison). The Shannon filter was made according to (Eqn. 6) with t - *tau_slide* to translate the window and width parameter *w* = 0.1. The window parameters were set to have approximately the same window width for each type window.

Sec. III. Algorithm Implementation and Development – Part II

In **Part 2.1** in **Appendix B. MATLAB codes**, the **music1.wav and music2.wave** files were loaded, with the former file being a piano recording and the latter a recorder recording. The time and frequency domains were discretized in a similar fashion as mentioned above with the frequency components adjusted for an even number of Fourier modes. The Gaussian filter was used as the filter window (width parameter = 100 and translation parameter = 0.1) and the raw spectrograms of the two music files were plotted to see and compare their base notes and overtones. In order to remove the overtones for each instrument, for each time frame, the Gábor filter was multiplied by the raw signal, Fourier transformed, and the maximum frequency amplitude and its index was found. The index was used to find the central frequency at that time frame and used another Gaussian filter, now in the frequency domain, to attenuate frequencies away from the central frequency. The overtone filtered data was then plotted as spectrograms using *pcolor()*.

Sec. IV. Computational Results – Part I

The spectrogram of *Handel's Messiah* could be made using Gábor filtering (Fig. 1). The accuracy of the spectrogram was checked by matching the highest frequency amplitudes (approx. 1000 and 500 Hz) to the higher color gradient level. The spectrogram also exhibited good time resolution as noted by the dark bands at approx. t = 2.5, 4.5, 5.5, and 6.8 seconds, which correlate to the time points of the lowest audio signal amplitudes.

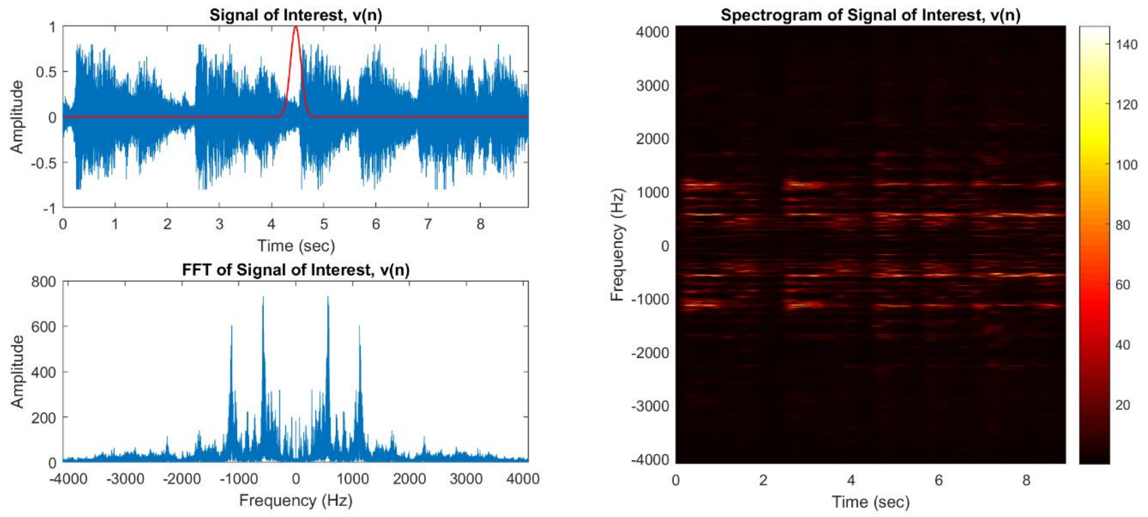


Figure 1. (*top left*) Audio signal of a portion of *Handel's Messiah* (blue) with a Gaussian filter window (red) at the middle time frame. (*bottom left*) Frequency spectrum of the portion of music. (*right*) Spectrogram of the portion of *Handel's Messiah*.

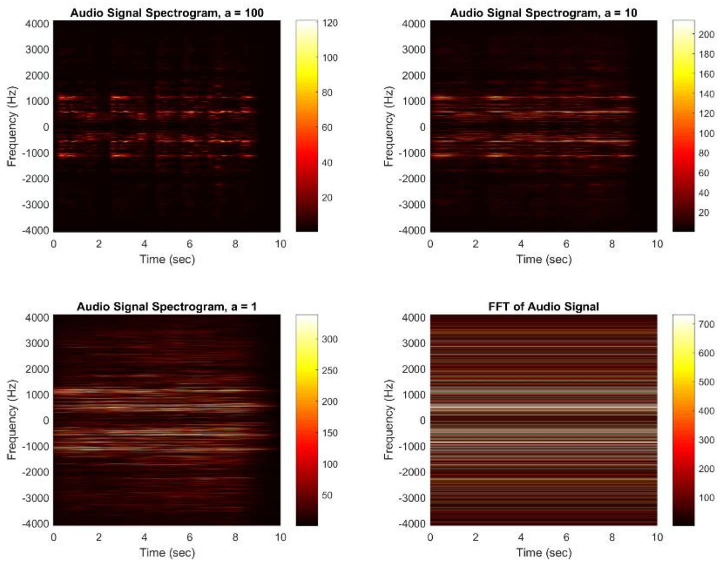


Figure 2. Spectrograms of a portion of *Handel's Messiah* with window with a = 100 (*top left*), a = 10 (*top right*), and a = 1 (*bottom left*). (*bottom right*) All frequencies and relative amplitude in audio signal (maximum frequency resolution) is shown.

The filter window width is inversely proportional to the width parameter a. Therefore, according to the Heisenberg uncertainty, as a decreases, the window width also increases, leading to better frequency resolution but worse time resolution. This is clearly seen in Fig. 2, where the greater window widths are not able to distinguish when there are low amplitudes of frequencies. By adjusting the translation parameter τ, the undersampling spectrogram showed poor time and frequency resolutions while the oversampling spectrograms show good time and frequency resolutions as the filter window can capture most of the frequency in small time frames (Fig. 3).

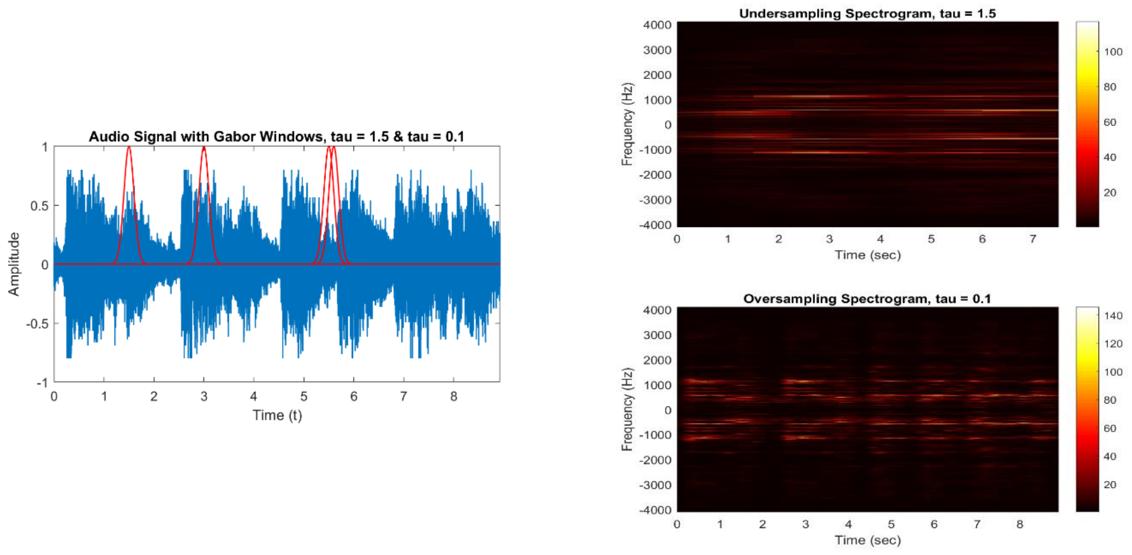
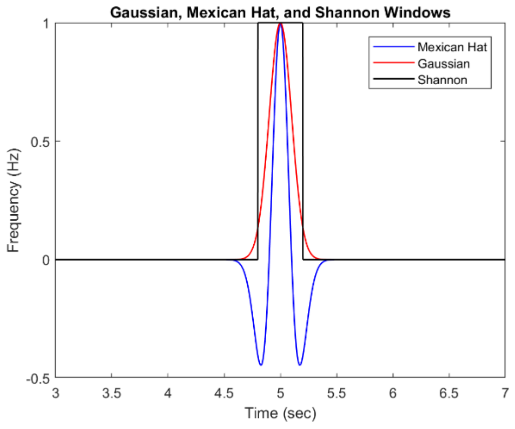


Figure 3. (*top left*) Audio signal of a portion of *Handel's Messiah* (blue) with Gábor windows showing undersampling and oversampling (red). (*top right*) Spectrogram of the undersampled signal with $\tau = 1.5$ and (*bottom right*) spectrogram of the oversampled signal with $\tau = 0.1$.



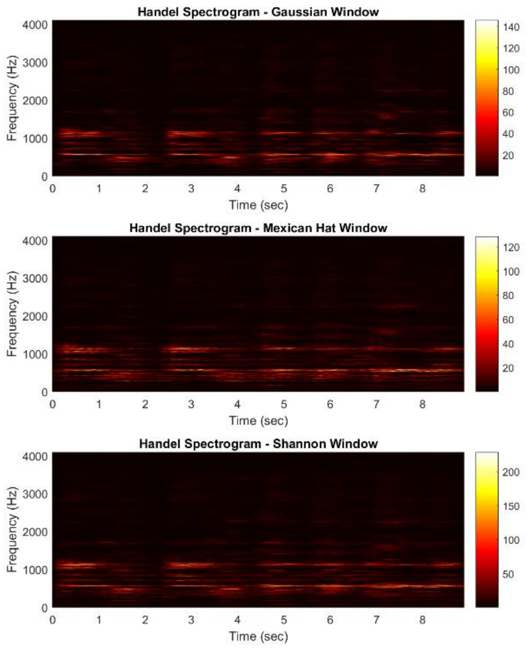
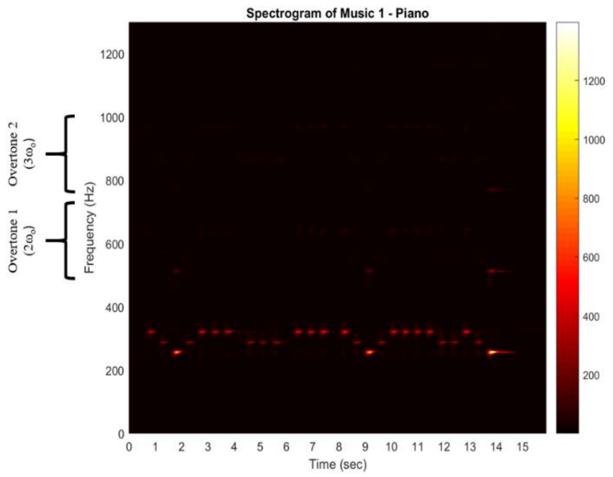


Figure 4. (*left*) Graph of the Mexican Hat, Gaussian, and Shannon windows with similar widths (~1 sec) and same height of 1. (*right*) Spectrograms of the signal filtered using the Gaussian (*top*) Mexican Hat (*middle*), and Shannon (*bottom*) windows – positive frequencies shown only.

As seen in Fig. 4, all the three window types were able to produce a sufficient spectrogram with relatively good and similar time and frequency resolutions. By adjusting each of the filter window parameters, the filters were optimized. Comparing the three windows, the Mexican hat is able to get high time resolution due to its fast decay. However, the two negative curves may attenuate some of the frequencies as the window translate. The Shannon window may seem to show a bit coarser frequency localization due to its steep decay rate.

Sec. IV. Computational Results – Part II



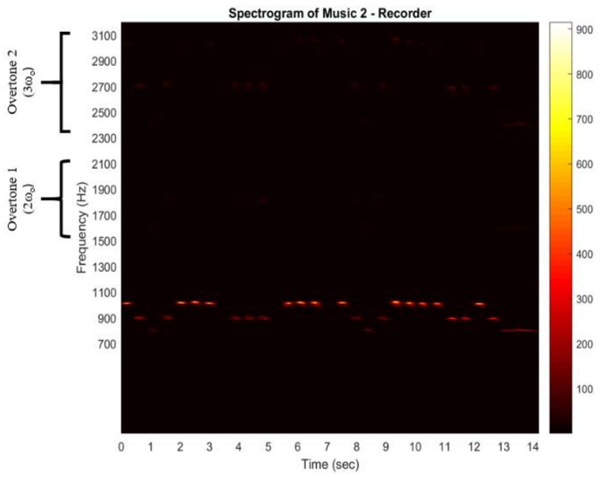


Figure 5. (*left*) Spectrogram of *Marry had a little lamb* on the piano. The two overtones above the base notes are shown. (*right*) Spectrogram of *Marry had a little lamb* on the recorder. The two overtones above the base notes are shown.

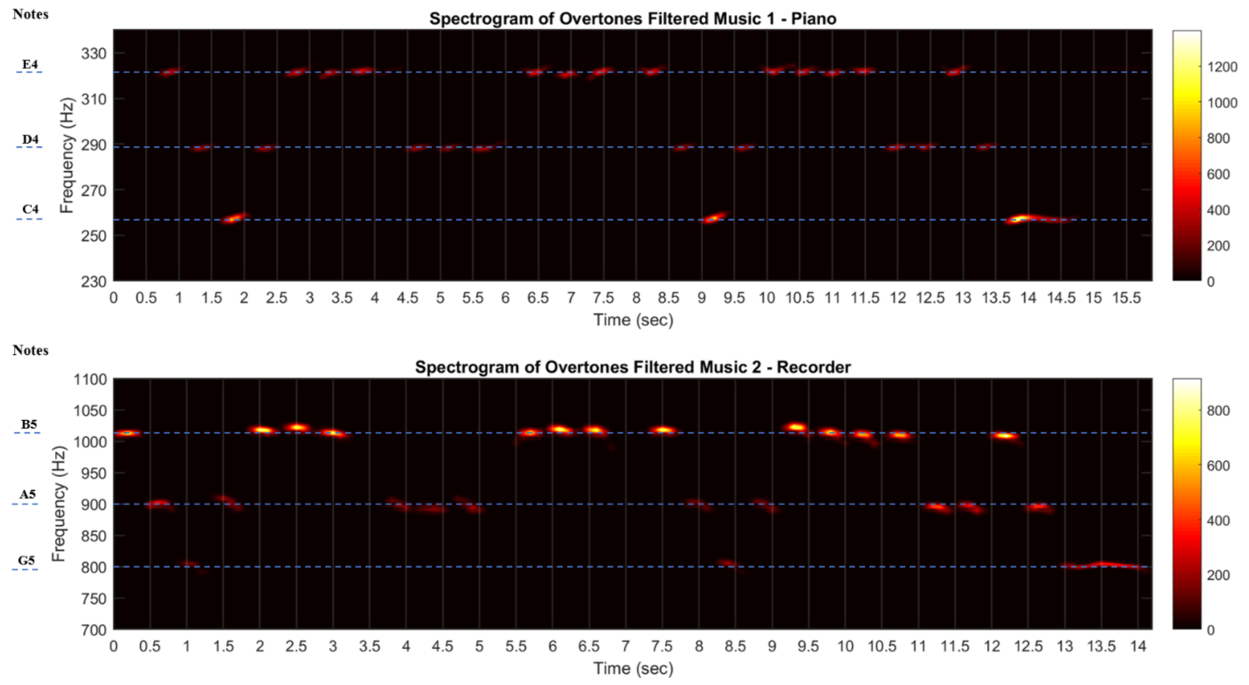


Figure 6. (*top*) Spectrogram of *Marry had a little lamb* on the piano. (*bottom*) Spectrogram of *Marry had a little lamb* on the recorder. The overtone filtered base notes with their respective notes, duration, and progression are shown.

From Fig. 5, the first and second overtones for the piano are more equally represented whereas the second overtone seems to be much more distinct than the first overtone. This change in overtone gives each instrument their own unique timbre. As seen in Fig. 6, the notes, duration, and progression of each note was plotted. E4 (~330 Hz), D4 (~294 Hz), C4 (~261 Hz), B5 (~330 Hz), A5 (~294 Hz), and G5 (~261 Hz) are the actual frequencies which were roughly around the frequencies found in the spectrogram. Each quarter notes are about 0.5 seconds given the bpm of this recording is around 120-130.

Sec. V. Summary and Conclusions

All in all, time-frequency analysis was carried out using the Gábor Transform. This allowed us to make spectrograms with varying frequency and time resolutions based on changing the width of the window, translation parameter, and filter type. Moreover, through solving this assignment, the importance and capabilities of Gábor Transforms could be exemplified along with how it can be used to gain frequency information at given time points. Furthermore, such transform allowed us to deduce the musical score of *Marry had a little lamb*.

Appendix A. MATLAB functions used and brief implementation explanation

- **fftshift():** shift the zero-frequency component to the center of the spectrum.
- **fft():** takes the discrete Fourier Transform.
- **pcolor():** makes a pseudocolor or “checkerboard” plot of input maxtrix on inputted x and y axis vectors or matrices.

Appendix B. MATLAB codes

```
%% AMATH 482 Homework 2
% Seong Hyun Han
% 2/07/20

%% Part 1.1
% Starter code - Plot a portion of Handel's Messiah

clear; close all; clc
load handel
% y = change in amplitude (loudness) of the sound wave with respect to time
v = y'; % Store transposed audio data (y)
% Fs = sample rate; length(v) = total sample size
figure(1)
subplot(2,2,1)
plot((1:length(v))/Fs,v);
set(gca,'Xlim',[0,length(v)/Fs],'Fontsize',10)
xlabel('Time (sec)');
ylabel('Amplitude');
title('Signal of Interest, v(n)');
hold on

% Playback the portion of music
% p8 = audioplayer(v,Fs);
% playblocking(p8);

% Produce spectrograms of the portion of music using Gabor filtering
% Implementation of the Gabor transform used in class

duration = length(v)/Fs; % Length of music (8.9249 sec)
L=duration; % Computational time domain
n=length(v); % Number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % Define the time domain discretization
t=t2(1:n); % Only consider the first n points for periodicity
% Fourier components rescaled to have frequency in hertz
k=(1/L)*[0:(n-1)/2 (-n+1)/2:-1]; % account for n being odd
ks=fftshift(k); % Fourier components with zero at the center

% Plot the Gaussian window on the audio signal
tau_ex = L/2;
a = 50;
g = exp(-a*(t-tau_ex).^2);
plot(t,g,'r','Linewidth',1)

% Frequency plot of the signal of interest
subplot(2,2,3)
plot(ks, fftshift(abs(fft(v))))
xlabel('Frequency (Hz)');
ylabel('Amplitude');
title('FFT of Signal of Interest, v(n)');
set(gca,'Xlim',[-abs(max(ks)), abs(max(ks))],'Fontsize',10)
% saveas(figure(2),'AMATH482_HW2_fig2.png');
% print(gcf,'AMATH482_HW2_fig2.png','-dpng','-r600');

% Create Gabor transform spectrogram
a1 = 50; % width parameter
tslide=0:0.1:L; % tau = 0.1
vgt_spec_g = zeros(length(tslide),n); % store filtered frequency data
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a1*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    vg=gabor.*v; % apply filter to signal (mutiplication in time domain)
    vgt=fft(vg);
    vgt_spec_g(j,:) = fftshift(abs(vgt)); % We don't want to scale it
end

% Plot spectrogram
subplot(2,2,[2,4])
pcolor(tslide,ks,vgt_spec_g.')
shading interp
```

```matlab
set(gca,'Fontsize',10)
colormap(hot)
colorbar
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Spectrogram of Signal of   Interest, v(n)');

%% Part 1.2
% Effect of window width of the Gabor transform on the spectrogram

figure(2)
% Spectrograms for varying a (window width) of the Gabor transform
a_vec = [100 10 1]; % a = 100, a = 10, a = 1
for jj = 1:length(a_vec)
    a = a_vec(jj);
    tslide=0:0.1:10;
    vgt_spec = zeros(length(tslide),n);
    for j=1:length(tslide)
        g=exp(-a*(t-tslide(j)).^2);
        vg=g.*v;
        vgt=fft(vg);
        vgt_spec(j,:) = fftshift(abs(vgt));
    end

    subplot(2,2,jj)
    pcolor(tslide,ks,vgt_spec.'),
    shading interp
    title(['Audio Signal Spectrogram, a = ',num2str(a)],'Fontsize',10)
    set(gca,'Fontsize',10)
    colormap(hot)
    colorbar
    xlabel('Time (sec)');
    ylabel('Frequency (Hz)');

end

% Plot unfiltered Fourier transformed audio data v
vgt_spec = repmat(fftshift(abs(fft(v))),length(tslide),1); % format to 90 by length(v) matrix
subplot(2,2,4)
pcolor(tslide,ks,vgt_spec.'),
shading interp
title('FFT of Audio Signal','Fontsize',10)
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
set(gca,'Fontsize',10)
colormap(hot)
colorbar

%% Part 1.3
% Effect of oversampling versus undersampling on the spectrogram

% Check to see if there is window overlap or not
figure(3)
tau_ex_1 = 1.5;
tau_ex_2 = 3;
tau_ex_3 = 5.5;
tau_ex_4 = 5.6;
a = 50;
g = exp(-a*(t-tau_ex_1).^2);
g2 = exp(-a*(t-tau_ex_2).^2);
g3 = exp(-a*(t-tau_ex_3).^2);
g4 = exp(-a*(t-tau_ex_4).^2);
plot(t,v,'Color',[0, 0.4470, 0.7410],'Linewidth',1)
hold on
plot(t,g,'r',t,g2,'r','Linewidth',1)
plot(t,g3,'r',t,g4,'r','Linewidth',1)
set(gca,'Xlim',[0 L],'Fontsize',10), xlabel('Time (t)'), ylabel('Amplitude')
title('Audio Signal with Gabor Windows, tau = 1.5 & tau = 0.1');

% Undersampling (windows don't overlap)
a1 = 50; % width parameter
```

```
tslide=0:1.5:L; % tau = 1.5
vgt_spec = zeros(length(tslide),n);
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a1*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    vg=gabor.*v; % apply filter to signal (mutiplication in time domain)
    vgt=fft(vg);
    vgt_spec(j,:) = fftshift(abs(vgt)); % We don't want to scale it
end

% Plot spectrogram
figure(4)
subplot(2,1,1)
pcolor(tslide,ks,vgt_spec.'),
shading interp
set(gca,'Fontsize',10)
colormap(hot)
colorbar
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Undersampling Spectrogram, tau = 1.5');

% oversampling (windows overlap)
a1 = 50; % width parameter
tslide=0:0.1:L; % tau = 0.1
vgt_spec = zeros(length(tslide),n);
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a1*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    vg=gabor.*v; % apply filter to signal (mutiplication in time domain)
    vgt=fft(vg);
    vgt_spec(j,:) = fftshift(abs(vgt)); % We don't want to scale it
end

% Plot spectrogram
subplot(2,1,2)
pcolor(tslide,ks,vgt_spec.'),
shading interp
set(gca,'Fontsize',10)
colormap(hot)
colorbar
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Oversampling Spectrogram, tau = 0.1');

%% Part 1.4
% Comparison of different Gabor windows: Gaussian window, Mexican hat
% wavelet, and step-function (Shannon) window

% Mexican hat
% Plot of Mexican hat wavelet in time
tau = 5; % filter center point
sigma = 0.1; % width parameter
c1 = (2/(sqrt(3*sigma)*pi.^(1/4))); % window (height) scaling parameter
c = 1; % scaling factor equal to Gaussian window and Shannon window for comparison
% Mexican hat equation
mexican_hat_ex = c*(1-((t-tau)/sigma).^2).*exp((-(t-tau).^2)/(2*(sigma.^2)));
figure(5)
plot(t, mexican_hat_ex, 'b', 'Linewidth',1)

tau_slide=0:0.1:L; % tau = 0.1
vmt_spec_m = zeros(length(tau_slide),n);
for j=1:length(tau_slide)
    % Mexican hat wavelet function / window
    mexican_hat = c*(1-((t-tau_slide(j))/sigma).^2).*exp((-(t-tau_slide(j)).^2)/(2*(sigma.^2)));
    vm=mexican_hat.*v; % apply filter to signal (mutiplication in time domain)
    vmt=fft(vm);
    vmt_spec_m(j,:) = fftshift(abs(vmt)); % We don't want to scale it
end

% Plot spectrogram using Mexican hat window
```

```matlab
figure(6)
pcolor(tau_slide,ks,vmt_spec_m.'),
shading interp
set(gca,'Fontsize',10)
colormap(hot)
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Handel Spectrogram - Mexican Hat Window');
colorbar

% Shannon filter
w = 0.4; % width parameter
tau_ex_3 = 5;
% Step function created using two heaviside functions
step_ex = heaviside((t-(tau_ex_3)+(w/2))) - heaviside((t-(tau_ex_3)-(w/2)));
figure(7)
plot(t, step)

tau_slide=0:0.1:L; % tau = 0.1
vmt_spec_s = zeros(length(tau_slide),n);
for j=1:length(tau_slide)
    % Shannon filter function / window
    step = heaviside((t-(tau_slide(j))+(w/2))) - heaviside((t-(tau_slide(j))-(w/2)));
    vm=step.*v; % apply filter to signal (mutiplication in time domain)
    vmt=fft(vm);
    vmt_spec_s(j,:) = fftshift(abs(vmt)); % We don't want to scale it
end

% Plot spectrogram using Shannon window
figure(8)
pcolor(tau_slide,ks,vmt_spec_s.'),
shading interp
set(gca,'Fontsize',12)
colormap(hot)
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Handel Spectrogram - Shannon Window');
colorbar

% Plot the three spectrograms (three windows)
figure(9)
subplot(3,1,1)
pcolor(tslide,ks,vgt_spec_g.')
shading interp
set(gca,'Ylim',[0, max(abs(ks))],'Fontsize',10)
colormap(hot)
colorbar
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Handel Spectrogram - Gaussian Window');

subplot(3,1,2)
pcolor(tau_slide,ks,vmt_spec_m.'),
shading interp
set(gca,'Ylim',[0, max(abs(ks))],'Fontsize',10)
colormap(hot)
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Handel Spectrogram - Mexican Hat Window');
colorbar

subplot(3,1,3)
pcolor(tau_slide,ks,vmt_spec_s.'),
shading interp
set(gca,'Ylim',[0, max(abs(ks))],'Fontsize',10)
colormap(hot)
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Handel Spectrogram - Shannon Window');
colorbar
```

```
% Plot the three windows
figure(10)
tau_ex_3 = 5;
g3 = exp(-a*(t-tau_ex_3).^2);
plot(t, mexican_hat_ex,'b','Linewidth',1)
hold on
plot(t,g3,'r','Linewidth',1)
plot(t, step_ex,'k','Linewidth',1)
hold off
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Gaussian, Mexican Hat, and Shannon Windows');
set(gca,'Xlim',[(5)-2, (5)+2],'Fontsize',10)
legend('Mexican Hat','Gaussian', 'Shannon')

%% Part 2.1
% Piano
figure(11)
[y2,Fs] = audioread('music1.wav');
p = y2'; % Store transposed audio data (y)
tr_piano=length(p)/Fs; % record time in seconds
plot((1:length(p))/Fs,p);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (piano)');
% p8 = audioplayer(y,Fs); playblocking(p8);

L=tr_piano; % computational time domain
n=length(p); % number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % define the time domain discretization
t=t2(1:n); % only consider the first n points for periodicity
k=((1)/L)*[0:n/2-1 -n/2:-1]; % Fourier components rescaled to have frequency in hertz
ks=fftshift(k); % Fourier components with zero at the center

% Frequency plot of the music (piano)
% figure(11)
% plot(ks, fftshift(abs(fft(p))))
% xlabel('Frequency(Hz)');
% ylabel('Amplitude');
% title('FFT of Music 1 - Piano');

a2 = 100; % width parameter
tslide=0:0.1:L; % tau = 0.1
pgt_spec = zeros(length(tslide),n);
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a2*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    pg=gabor.*p; % apply filter to signal (mutiplication in time domain)
    pgt=fft(pg);
    pgt_spec(j,:) = fftshift(abs(pgt)); % We don't want to scale it
end

% Plot Music 1 piano spectrogram
figure(12)
subplot(1,2,1)
pcolor(tslide,ks,pgt_spec.'),
xticks(0:1:tr_piano)
yticks(0:200:1300)
shading interp
set(gca,'Ylim',[0 1300],'Fontsize',10)
colormap(hot)
colorbar
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Spectrogram of Music 1 - Piano');

% Recorder

% figure(13)
[y3,Fs] = audioread('music2.wav');
r = y3'; % Store transposed audio data (y)
tr_rec=length(r)/Fs; % record time in seconds
```

```
% plot((1:length(r))/Fs,r);
% xlabel('Time [sec]'); ylabel('Amplitude');
% title('Mary had a little lamb (recorder)');
% p8 = audioplayer(y,Fs); playblocking(p8);

L=tr_rec; % computational time domain
n=length(r); % number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % define the time domain discretization
t=t2(1:n); % only consider the first n points for periodicity
k=(1/L)*[0:n/2-1 -n/2:-1]; % Fourier components rescaled to have frequency in hertz
ks=fftshift(k); % Fourier components with zero at the center

% Frequency plot of the music (recorder)
% figure(14)
% plot(ks, fftshift(abs(fft(r))))
% xlabel('Frequency(Hz)');
% ylabel('Amplitude');
% title('FFT of Music 2 - Recorder');

a3 = 100; % width parameter
tslide=0:0.1:L; % tau = 0.1
rgt_spec = zeros(length(tslide),n);
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a3*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    rg=gabor.*r; % apply filter to signal (mutiplication in time domain)
    rgt=fft(rg);
    rgt_spec(j,:) = fftshift(abs(rgt)); % We don't want to scale it
end

% Plot Music 2 spectrogram
subplot(1,2,2)
pcolor(tslide,ks,rgt_spec.'),
xticks(0:1:tr_piano)
yticks(700:200:3200)
shading interp
set(gca,'Ylim',[0 3200],'Fontsize',10)
colormap(hot)
colorbar
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Spectrogram of Music 2 - Recorder');

%% Part 2.2
% Overtone Remover
% Piano
clear all; close all; clc
figure(13)
[y2,Fs] = audioread('music1.wav');
p = y2'; % Store transposed audio data (y)
tr_piano=length(p)/Fs; % record time in seconds
plot((1:length(p))/Fs,p);
set(gca, 'YGrid', 'off', 'XGrid', 'on')
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (piano)');
% p8 = audioplayer(y,Fs); playblocking(p8);

L=tr_piano; % computational time domain
n=length(p); % number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % define the time domain discretization
t=t2(1:n); % only consider the first n points for periodicity
k=(1/L)*[0:n/2-1 -n/2:-1]; % Fourier components rescaled to have frequency in hertz
ks=fftshift(k); % Fourier components with zero at the center

% Frequency plot of the music (recorder)
% figure(14)
% plot(ks, fftshift(abs(fft(r))))
% xlabel('Frequency(Hz)');
% ylabel('Amplitude');
% title('FFT of Music 2 - Recorder');
```

```
a3 = 100; % width parameter
tslide=0:0.1:L;
pgt_spec = zeros(length(tslide),n);
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a3.*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    pg=gabor.*p; % apply filter to signal (mutiplication in time domain)
    pgt=fft(pg);
    [M,I] = max(abs(pgt)); % find maximum frequency amplitude value and its index
    center_freq = k(I); % define center frequency from index
    % width parameter = 1
    overtone = exp(-(0.5).*(k-center_freq).^2); % use Gaussian filter around center frequency
    pgt_final = pgt.*overtone; % apply Gaussian filter in Fourier domain
    pgt_spec(j,:) = fftshift(abs(pgt_final)); % We don't want to scale it
end

% Plot spectrogram
figure(15)
subplot(2,1,1)
pcolor(tslide,ks,pgt_spec.'),
xticks(0:0.5:tr_piano)
yticks(230:20:340)
shading interp
set(gca,'Ylim',[230 340],'Fontsize',10)
colormap(hot)
colorbar
grid on
set(gca,'layer','top')
ax = gca;
ax.GridColor = 'w';
ax.LineWidth = 1;
ax.YGrid = 'off';
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Spectrogram of Overtones Filtered Music 1 - Piano');
colorbar

% Recorder
%clear all; close all; clc
%figure(13)
[y3,Fs] = audioread('music2.wav');
r = y3'; % Store transposed audio data (y)
tr_rec=length(r)/Fs; % record time in seconds
%plot((1:length(r))/Fs,r);
%xlabel('Time [sec]'); ylabel('Amplitude');
%title('Mary had a little lamb (recorder)');
% p8 = audioplayer(y,Fs); playblocking(p8);

L=tr_rec; % computational time domain
n=length(r); % number of Fourier modes (2^n)
t2=linspace(0,L,n+1); % define the time domain discretization
t=t2(1:n); % only consider the first n points for periodicity
k=(1/L)*[0:n/2-1 -n/2:-1]; % Fourier components rescaled to have frequency in hertz
ks=fftshift(k); % Fourier components with zero at the center

% Frequency plot of the music (recorder)
% figure(14)
% plot(ks, fftshift(abs(fft(r))))
% xlabel('Frequency(Hz)');
% ylabel('Amplitude');
% title('FFT of Music 2 - Recorder');

a3 = 100; % width parameter
tslide=0:0.1:L;
rgt_spec = zeros(length(tslide),n);
for j=1:length(tslide)
    % Gabor filter function / window
    gabor=exp(-a3.*(t-tslide(j)).^2); % tau = tslide(j) = translation parameter
    rg=gabor.*r; % apply filter to signal (mutiplication in time domain)
    rgt=fft(rg);
    [M,I] = max(abs(rgt));
    center_freq = k(I);
    % width parameter = 1
    overtone = exp(-(0.01).*(k-center_freq).^2);
    rgt_final = rgt.*overtone;
    rgt_spec(j,:) = fftshift(abs(rgt_final)); % We don't want to scale it
end

% Plot spectrogram
subplot(2,1,2)
pcolor(tslide,ks,rgt_spec.'),
xticks(0:0.5:tr_rec)
yticks(700:50:1100)
shading interp
set(gca,'Ylim',[700 1100],'Fontsize',10)
colormap(hot)
colorbar
grid on
set(gca,'layer','top')
ax = gca;
ax.GridColor = 'w';
ax.LineWidth = 1;
ax.YGrid = 'off';
xlabel('Time (sec)');
ylabel('Frequency (Hz)');
title('Spectrogram of Overtones Filtered Music 2 - Recorder');
```

References

1. Kutz, Jose Nathan. *Data-Driven Modeling &amp; Scientific Computation: Methods for Complex Systems &amp; Big Data*. Oxford University Press, 2013.
2. Mexican hat wavelet. (2019, December 03). Retrieved from https://en.wikipedia.org/wiki/Mexican_hat_wavelet