AMATH 482 Homework 2

Rishabh Verma

February 10th, 2021

Abstract

1 Introduction and Overview

The MNIST dataset contains labeled scans of handwritten digits. Each scan is a 28x28 pixel image, and each pixel is an 8-bit greyscale value. Each image contains a size-normalized centered digit. There are 60,000 scans in the training set and 10,000 scans in the testing set.

2 Theoretical Background

2.1 Limitations of Discrete Fourier Transform

The Discrete Fourier Transform is a method to decompose a signal into its constituent frequencies. It is given by the formula for all bins $k = -N/2, ..., N/2 - 1$:

$$\hat{x}_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}} \tag{1}$$

where x_n represents a signal. The DFT is subject to aliasing, in that

$$\hat{x}_k = \hat{x}_{k+\mu k} \text{ for all integers } \mu, \tag{2}$$

and this will be used to show the following result. If a periodic signal x_n is shifted in time, i.e. there exists some non-zero integer τ and a shifted signal $y_n = x_{n-\tau}$, then

$$\begin{aligned}
\hat{y}_k &= \frac{1}{N}\sum_{n=0}^{N-1} y_n e^{\frac{2\pi ikn}{N}} = \frac{1}{N}\sum_{n=0}^{N-1} x_{n-\tau} e^{\frac{2\pi ikn}{N}} \\
&= \frac{1}{N}\sum_{n=\tau}^{N-1+\tau} x_n e^{\frac{2\pi ik(n+\tau)}{N}} \text{ using periodicity of } x_n \\
&= \frac{1}{N}\sum_{n=\tau}^{N-1} x_n e^{\frac{2\pi ik(n+\tau)}{N}} + \frac{1}{N}\sum_{n=N}^{N-1+\tau} x_n e^{\frac{2\pi ik(n+\tau)}{N}} \\
&= \frac{1}{N}\sum_{n=\tau}^{N-1} x_n e^{\frac{2\pi ik(n+\tau)}{N}} + \frac{1}{N}\sum_{n=0}^{\tau-1} x_n e^{\frac{2\pi ik(n+\tau)}{N}} \text{ using aliasing} \\
&= \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{\frac{2\pi ik(n+\tau)}{N}},
\end{aligned} \tag{3}$$

and so the result is simply subject to a phase change in the exponential factor. This means that the output of a DFT is unable to detect when a frequency occurs in the signal.

2.2 The Gabor Transform

For the purposes of this section, I will refer only to the Discrete Gabor Transform.

2.2.1 Frequency-time analysis

The Gabor Transform operates on a very simple premise. Multiply the signal by a symmetric windowing function centered at some time τ, and then compute the DFT to determine the frequency composition at that time. The bandwidth of the windowing function is important. If the window is too narrow, a precise measurement of frequency cannot be made. If the window is too wide, the output of the DFT will not be localized in time, and we run into the same problem that the FFT experiences in Equation 3: it will be impossible to discern when a frequency occurs. This is known as the acoustic Heisenberg uncertainty principle.

2.2.2 Application to transcribing

With a fitting choice of window width, the Gabor Transform can be computed at various time instants τ, forming snapshots of the frequency composition. For the data at hand, this means that all frequencies emitted by all instruments will be captured. Most instruments, particularly distorted electric guitars, have a rich harmonic spectrum in which the fundamental frequency is accompanied by many overtones. A musician looking to transcribe a song is concerned with an instrument's pitch, which usually corresponding to the fundamental frequency. With some information on the harmonic content of the instrument we are looking to transcribe, we can use filtering to manipulate the harmonic spectrum and emphasize the fundamental frequency of each note. The Gabor transform can then identify the dominant frequency being played at time τ, corresponding with the note being played at τ.

2.2.3 Filter choice

It is important to note that the filter used for a Gabor Transform must be symmetric, and should by convention have unity L2-power. The filters used in this paper are Gaussian filters of height 1 and varying bandwidth, so they are symmetric, but their L2-power is not guaranteed to be unity. Regardless, all audio-samples are normalized to have a max amplitude of ± 1 before and after processing. This prioritizes music production conventions over signal processing conventions.

2.3 Frequency manipulation

2.3.1 Overtones

An instrument's overtones usually occur in integer multiples above the fundamental frequency, never below, and they tend to decay in amplitude. I would like for this spectral decay to be more pronounced so that it may be easier to identify the fundamental frequency, so I can create a gentle low-pass filter which becomes stronger for increasing frequencies.

In a sample with a mix of instruments, it becomes necessary to isolate the instrument of interest. Unless the instrument has the lowest pitch (such as the bass guitar in Comfortably Numb), it will require a band-pass filter. Identifying the corresponding cutoffs requires trial and error, and careful spectrogram analysis.

2.3.2 Spectrogram

The output of the Gabor Transform is well-displayed via spectrogram. This has time on the horizontal axis, frequency on the vertical axis, and at all points between these axes, the signal power at that frequency and that time is plotted in color.

2.3.3 Scientific frequency vs musical pitch

A frequency can be converted into the note it most closely matches. This note can be displayed in standard scientific notation (e.g. `C4`), or it can be displayed in a manner corresponding to how the MIDI (Musical Instrument Digital Interface) protocol stores notes as 7-bit integers. The note `C4` is stored as 60 in base-10,

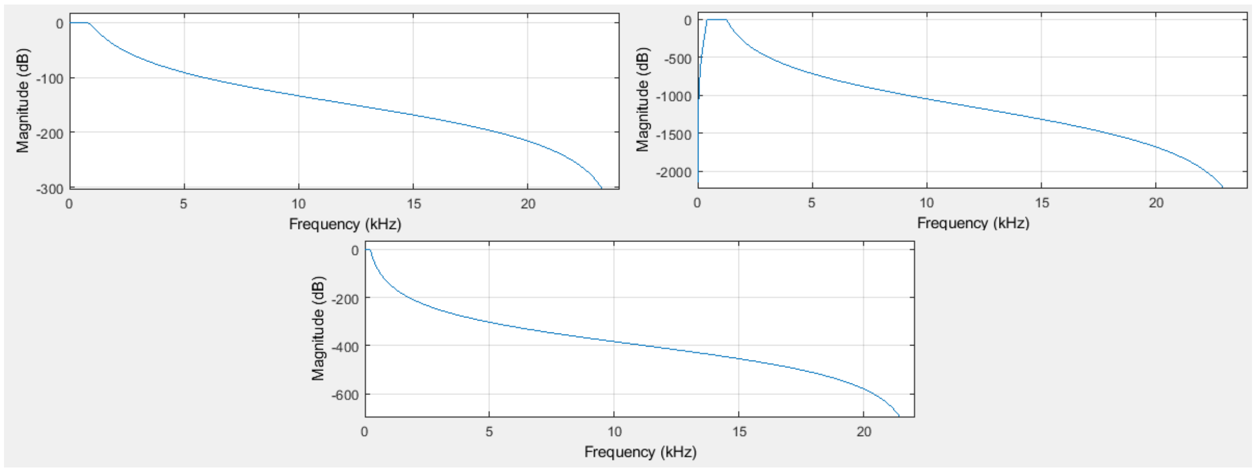

Figure 1: Left-to-right: The filters used for Sweet Child O Mine guitar, Comfortably Numb bass, Comfortably Numb solo

```
Algorithm 1: Gabor Transform($x_t, \alpha, \tau$)
    Initialize Gaussian function $g(t, \tau) = e^{a(t-\tau)^2}$
    Compute $y_t = x_t g(t, \tau)$
    Compute and return the DFT of $y_t$
```

and a change in semitones is stored as a corresponding change (e.g. `C5` is +12 semitones higher, so it is stored as 72). This is useful for easily plotting a transcription without regards for musical notation.

3 Algorithm Implementation and Development

3.1 Filtering

For the guitar in the Guns N Roses sample, I apply a weak low-pass filter to curb the overtones before transcribing.

For the bass in the Pink Floyd sample, I apply a strong low-pass filter to isolate the bass in the mix, particularly to avoid the rhythm guitar.

For the lead guitar in the Pink Floyd sample, I apply a strong band-pass filter to isolate the solo, particularly to avoid the overtones of the bass and rhythm guitar.

3.2 Transcribing

The process of transcription begins by computing the spectrogram with Algorithm 2. Then, the frequency bin with maximum power in each column is mapped to the corresponding note. Sequential instants with the same note are regarded as a single extended note (e.g. `[F#5, F#5, G5, D#5, D#5]` $\longrightarrow$ `[F#5, G5, D#5]`)

Filters are designed, generated, and applied via MATLAB's Signal Processing Toolbox.

The parameter T is calibrated with regards to performance and memory consumption. The parameter α is calibrated with regards to subsection 2.2.1.

See Appendix A for more subroutines written to clean the resulting output into a human-presentable form.

4 Computational Results

Algorithm 2: Spectrogram(x_t, α, T)

```
Initialize an empty solution matrix with dimensions of length($x_t$) and length($T$)
for all $\tau \in T$ do
  Store Gabor Transform($x_n, \alpha, \tau$) (Algorithm 1) as a column in the solution matrix
end for
Return the solution matrix.
```

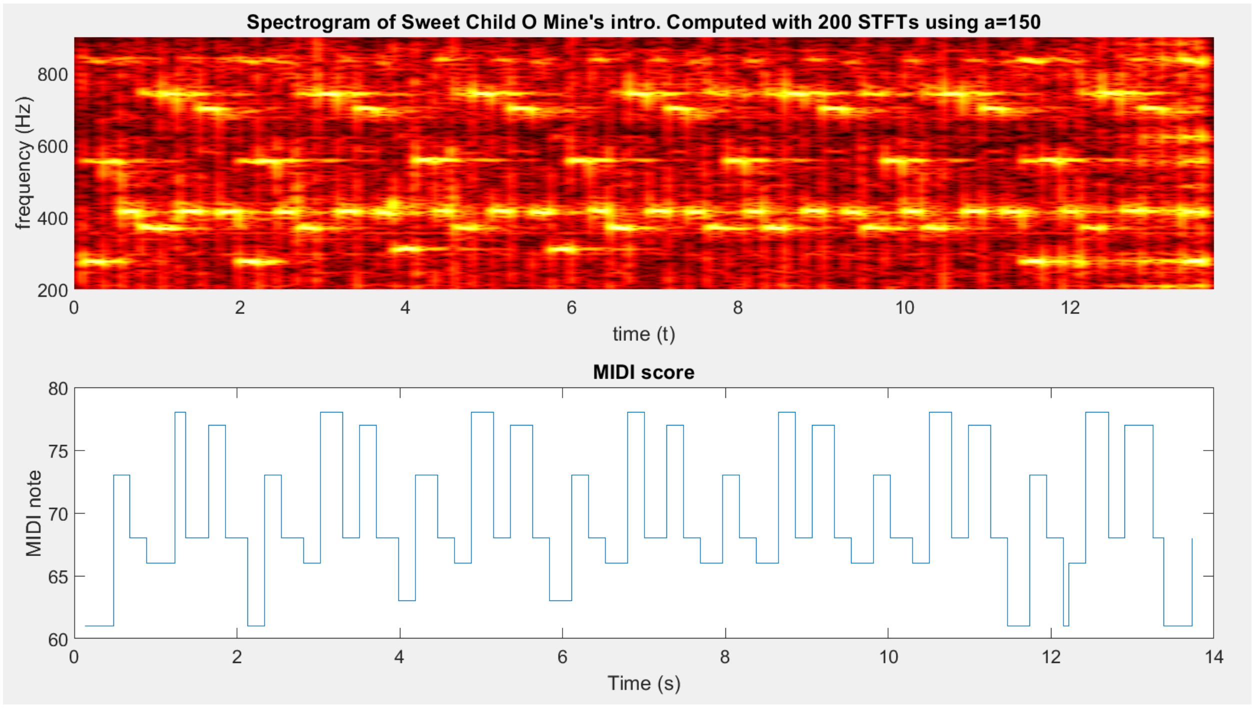

Figure 2: Analysis of Guns N Roses riff

5 Summary and Conclusions

The method of filtering and applying the Gabor transform can be useful for music transcription of isolated instruments. It works very well for the Guns N Roses riff, with perfect transcription up until the last two seconds where other instruments vamp in.

This method also works fairly well for the filtered Pink Floyd bassline, though this has some artifacts which can be seen as large oscillating jumps of semitones. This could possibly be improved with numerical optimization of the Gabor parameters to minimize ”jumpiness” in the score.

This method unfortunately does not work well with the Comfortably Numb solo. One reason could be because some of the notes are played very quickly, and the frequency-time analysis just can't get enough time resolution without sacrificing excessive frequency resolution. However, the transcription also does not seem to capture the long notes well despite my best efforts to adjust the filter cutoffs.

I conclude that the method of simply filtering and computing Gabor transform is too simplistic for dealing with a dense mix, though it does succeed in dealing with isolated instruments with dense harmonic spectrums as demonstrated with Guns N Roses.

One application could include a Digital Audio Workstation VST plugin which reads in isolated audio of an instrument playing, filters it, and transcribes the dominant frequencies. Because I am able to compute

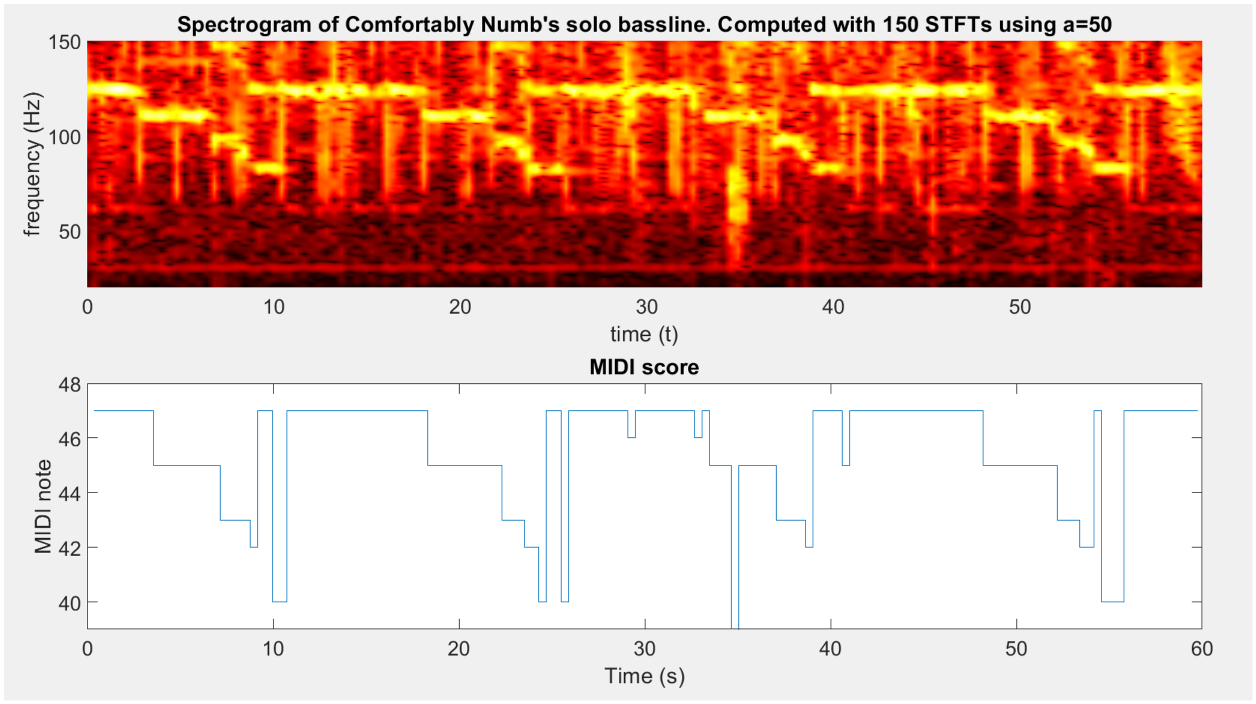

Figure 3: Analysis of low-pass filtered Pink Floyd bassline during solo

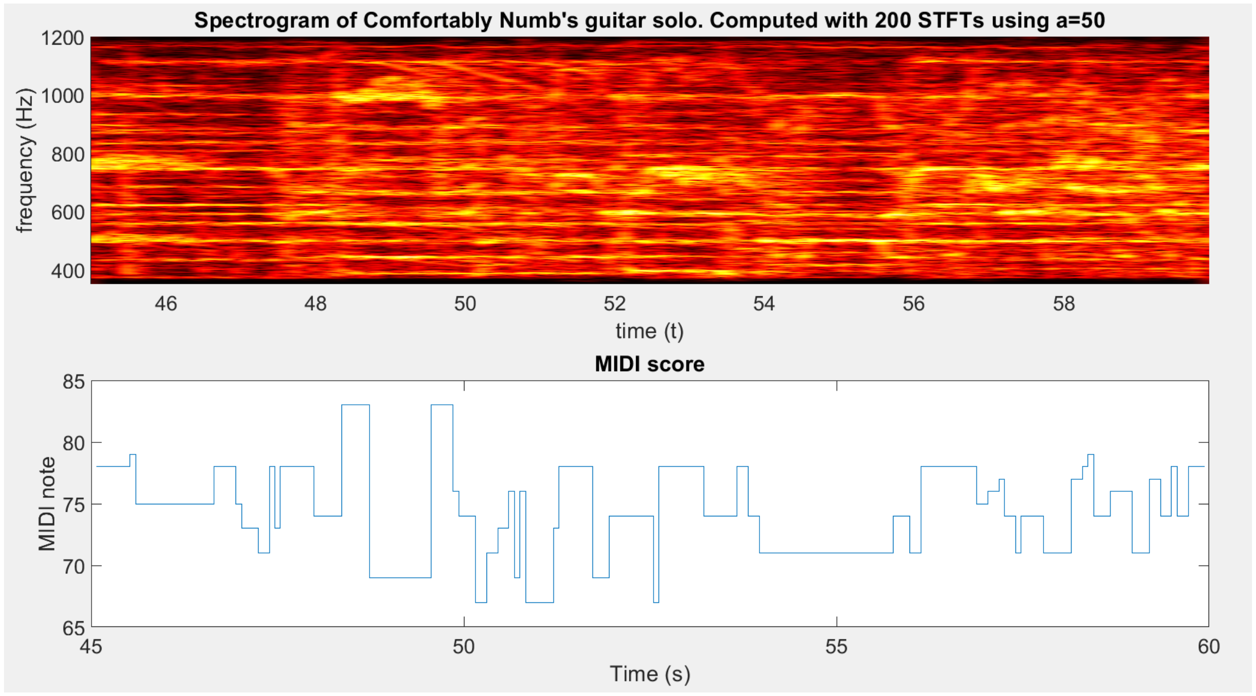

Figure 4: Analysis of last 15 seconds of band-pass filtered Pink Floyd guitar solo

a Gabor transform of 60 seconds of music in significantly less than 60 seconds, this can be implemented in almost real-time, though in order to compute a Gabor transform, a window might be needed which, unlike the Gaussian window, reaches zero; a Hamming window would do. In that case, near real-time music transcription could be implemented with a delay equal to the width of the window.

```
B2, A2, G2, F#2, B2, E2,
B2, A2, G2, F#2, E2, B2, E2,
B2, A#2, B2, A#2,
B2, A2, D2, A2, G2, F#2,
B2, A2, B2, A2, G2, F#2,
B2, E2, B2
```

(a) Sweet Child O Mine riff

```
C#4,C#5,G#4,F#4,F#5,G#4,F5,G#4,
C#4,C#5,G#4,F#4,F#5,G#4,F5,G#4,
D#4,C#5,G#4,F#4,F#5,G#4,F5,G#4,
D#4,C#5,G#4,F#4,F#5,G#4,F5,G#4,
F#4,C#5,G#4,F#4,F#5,G#4,F5,G#4,
F#4,C#5,G#4,F#4,F#5,G#4,F5,G#4,
C#4,C#5,G#4,C#4,F#4,F#5,G#4,F5,
G#4,C#4
```

(b) Comfortably Numb bass

```
F#5,G5,D#5,F#5,D#5,C#5,B4,F#5,
C#5,F#5,D5,B5,A4,B5,E5,D5,
G4,B4,C#5,E5,A4,E5,G4,C#5,
F#5,A4,D5,G4,F#5,D5,F#5,D5,
B4,D5,B4,F#5,D#5,E5,F5,D5,
B4,D5,B4,F5,F#5,G5,D5,E5,
B4,F5,D5,F#5,D5,F#5
```

(c) Comfortably Numb solo excerpt

Figure 5: Song transcriptions

Appendix A MATLAB Functions

- `note = freq2note(freq)` inputs a vector of frequencies. It computes the frequencies' relative pitch via modular arithmetic and relative octave via division, all relative to the base note `C0`, and outputs as a vector of strings.
- `note = freq2midi(freq)` inputs a vector of frequencies. It computes the number of semitones each frequency is above `C0`, and uses that to return the MIDI pitch values as a vector of integers.
- `notes = removeDuplicate(notes_raw)` inputs a vector of notes (as strings). If two or more adjacent notes have the same value, it quashes them into one note.
- The following methods respectively apply the filters designed for Sweet Child O Mine guitar, Comfortably Numb bass, and Comfortably Numb solo
  - `y_f = gnrlowpass48(y)`
  - `y_f = bassFilter(y)`
  - `y_f = soloFilter(y)`

Appendix B MATLAB Code

B.1 main.m

```
figure(1)
[y_gnr, Fs_gnr] = audioread('GNR.m4a');
y_gnr = y_gnr/max(abs(y_gnr));
tr_gnr=length(y_gnr)/Fs_gnr; % song duration in seconds
plot((1:length(y_gnr))/Fs_gnr,y_gnr);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Sweet Child O Mine intro');
% p8 = audioplayer(y_gnr, Fs_gnr); play(p8);
```

```
figure(2)
[y_pf, Fs_pf] = audioread('Floyd.m4a');
y_pf = y_pf/max(abs(y_pf));
y_pf = y_pf(1:end-1);
tr_pf=length(y_pf)/Fs_pf; % song duration in seconds
plot((1:length(y_pf))/Fs_pf,y_pf);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Comfortably Numb solo');
% p8 = audioplayer(y_pf, Fs_pf); playblocking(p8);

y_gnr = cast(y_gnr, 'single');
y_pf = cast(y_pf, 'single');

fontsize=20;

%% Let's score Guns N Roses

figure(3);
frequency_step = 8;  % only record every step-th bin to save memory
song_portion=1;  % portion of the song to display
a=150;
num_gabors = 200;

threshold = 3;  % if max < threshold at some time, we will record a rest
max_freq = 20000;  % to avoid higher notes while scoring

Fs = Fs_gnr;
y = y_gnr;
tr = tr_gnr;

y_f = gnrlowpass48(y);
% p8 = audioplayer(y_f, Fs); play(p8);

track = y_f(1:floor(end*song_portion));
sz = length(track);
tau = linspace(0, tr*song_portion, num_gabors);  % gabor centers, unit seconds

t = (1:sz)/Fs; % song sample time-values
t = cast(t, 'single');
Sgt_spec = cast(zeros(floor(sz/frequency_step),num_gabors), 'single');

gauss = @(t, tau) exp(-a*(t-tau).^2);
% Perform gabor transform, store in Sgt_spec
for j = 1:num_gabors
    g = gauss(t, tau(j));
    g = cast(reshape(g,sz,1), 'single');
    Sg = g.*track;
    Sgt = fft(Sg);
    Sgt = Sgt(1:frequency_step:end);
    % % (un)-comment the line below to resolve off-by-one errors
    Sgt = Sgt(1:end-1);
    Sgt_spec(:,j) = cast(fftshift(log(1+abs(Sgt))), 'single');
end
clear Sgt Sg g;
```

```
% Display spectrogram
subplot(2,1,1);
ks = linspace(-Fs/2, Fs/2, sz/frequency_step);
pcolor(tau, ks, Sgt_spec)
shading interp
set(gca, 'ylim', [200, 900], 'Fontsize', fontsize)
colormap(hot)
%colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
% title("First " + floor(song_portion*tr) + " seconds with " + num_gabors + " a = " + a + " STFTs");
title("Spectrogram of Sweet Child O Mine's intro. Computed with " + num_gabors + " STFTs using a=" + a)

% Let's get the notes.
bandwidth = (length(ks)/2)*max_freq/(Fs/2); % half bandwidth, measured as # of bins below max_freq
% for LPF centered at 0, to avoid transcribing higher notes
minbin = floor(length(ks)/2 - bandwidth);
maxbin = floor(length(ks)/2 + bandwidth);

freqs = zeros(1, num_gabors);
for j = 1:num_gabors
    [M,I] = max(Sgt_spec(minbin:maxbin,j)); I = I + minbin;
    % [M,I] = max(Sgt_spec(:,j));
    if M > threshold
        freqs(j) = abs(ks(I));
    else
        freqs(j) = 0;
    end
end
subplot(2,1,2);
stairs((1:length(freqs))*(song_portion*tr)/length(freqs), freq2midi(freqs));
xlabel("Time (s)");
ylabel("MIDI note");
title("MIDI score");
set(gca, 'Fontsize', fontsize);



%% Let's isolate the Floyd bassline. Let's start with LPF above 250 Hz
y_pf_f = y_pf;
y_pf_f = lowpassbass(y_pf_f);

sz = size(y_pf);
sz = sz(1);
ks = linspace(-Fs_pf/2, Fs_pf/2, sz);
% a=3e-6 sounds best, but too many overtones for analysis
a=1e-6;
gauss = @(k, f0) exp(-a*(k-f0).^2);
%p8 = audioplayer(y_pf_f, Fs_pf); play(p8);


freqs = [125 110 100 80]./2;
filter = zeros(1,length(y_pf_f));
for j = 1:length(freqs)
```

```
    freq = freqs(j);
    for order = 1:20
        filter = filter + 1/sqrt(order)*(gauss(ks, order*freq) + gauss(ks, -order*freq));
    end
end
filter = filter/max(filter);
filter = reshape(filter, sz, 1);

fy = fft(y_pf_f);
fy = fftshift(filter) .* fy;
y_pf_f = real(ifft(fy));

% y_pf_f = lowpass48(y_pf_f);

p8 = audioplayer(real(y_pf_f), Fs_pf); play(p8);

%% Now let's get a spectrogram and score the bass in y_pf.
figure(3);
frequency_step = 16;  % only record every step-th bin to save memory
song_portion=1;  % portion of the song to display
a=50;
num_gabors = 150;

Fs = Fs_pf;
y = y_pf_f;
tr = tr_pf;

threshold = 3;  % if max < threshold at some time, we will record a rest
max_freq = 130;  % to avoid higher notes while scoring

gauss = @(t, tau) exp(-a*(t-tau).^2);
track = y(1:floor(end*song_portion));
sz = length(track);
tau = linspace(0, tr*song_portion, num_gabors);  % gabor centers, unit seconds

t = (1:sz)/Fs; % song sample time-values
t = cast(t, 'single');
Sgt_spec = cast(zeros(floor(sz/frequency_step),num_gabors), 'single');

% Perform gabor transform, store in Sgt_spec
for j = 1:num_gabors
    g = gauss(t, tau(j));
    g = cast(reshape(g,sz,1), 'single');
    Sg = g.*track;
    Sgt = fft(Sg);
    Sgt = Sgt(1:frequency_step:end);
    % % (un)-comment the line below to resolve off-by-one errors
    % Sgt = Sgt(1:end-1);
    Sgt_spec(:,j) = cast(fftshift(log(1+abs(Sgt))), 'single');
end
clear Sgt Sg g;

% Display spectrogram
subplot(2,1,1);
```

```
ks = linspace(-Fs/2, Fs/2, sz/frequency_step + 1);
ks = ks(1:end-1);
pcolor(tau, ks, Sgt_spec)
shading interp
set(gca, 'ylim', [20, 150], 'Fontsize', fontsize)
colormap(hot)
% colorbar
xlabel('time (t)'), ylabel('frequency (Hz)');
title("Spectrogram of Comfortably Numb's solo bassline. Computed with " + num_gabors + " STFTs using a=

% Let's get the notes.
bandwidth = (length(ks)/2)*max_freq/(Fs/2); % half bandwidth, measured as # of bins below max_freq
% for LPF centered at 0
minbin = floor(length(ks)/2 - bandwidth);
maxbin = floor(length(ks)/2 + bandwidth);

notes = strings(1,num_gabors);
freqs = zeros(1, num_gabors);
for j = 1:num_gabors
    [M,I] = max(Sgt_spec(minbin:maxbin,j)); I = I + minbin;
    % [M,I] = max(Sgt_spec(:,j));
    if M > threshold
        freqs(j) = abs(ks(I));
    else
        freqs(j) = 0;
    end
end
subplot(2,1,2);
stairs((1:length(freqs))*(song_portion*tr_pf)/length(freqs), freq2midi(freqs));
xlabel("Time (s)");
ylabel("MIDI note");
ylim([39 48]);
title("MIDI score");
set(gca, 'Fontsize', fontsize);


%% And now let's try the Floyd solo
y = y_pf;
Fs = Fs_pf;
tr = tr_pf;
figure(4);
frequency_step = 16;  % only record every step-th bin to save memory
song_portion=0.25;  % portion of the song to display
a=50;
num_gabors = 200;

threshold = 4.2;  % if max < threshold at some time, we will record a rest
max_freq = 10000;  % to avoid higher notes while scoring

y_f = soloFilter(y);

% track = y_f(1:floor(end*song_portion));
track = y_f(floor(end*(1-song_portion)):end);
sz = length(track);
```

```
p8 = audioplayer(track, Fs); play(p8);
tau = linspace(0, tr*song_portion, num_gabors);  % gabor centers, unit seconds

t = (1:sz)/Fs; % song sample time-values
t = cast(t, 'single');
Sgt_spec = cast(zeros(floor(sz/frequency_step),num_gabors), 'single');

gauss = @(t, tau) exp(-a*(t-tau).^2);
% Perform gabor transform, store in Sgt_spec
for j = 1:num_gabors
    g = gauss(t, tau(j));
    g = cast(reshape(g,sz,1), 'single');
    Sg = g.*track;
    Sgt = fft(Sg);
    Sgt = Sgt(1:frequency_step:end);
    % % (un)-comment the line below to resolve off-by-one errors
    Sgt = Sgt(1:end-1);
    Sgt_spec(:,j) = cast(fftshift(log(1+abs(Sgt))), 'single');
end
clear Sgt Sg g;

% Display spectrogram
subplot(2,1,1);
ks = linspace(-Fs/2, Fs/2, sz/frequency_step);
pcolor(tau+45, ks, Sgt_spec)
shading interp
set(gca, 'ylim', [350, 1200], 'Fontsize', fontsize)
colormap(hot)
%colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
title("Spectrogram of Comfortably Numb's guitar solo. Computed with " + num_gabors + " STFTs using a="

% Let's get the notes.
bandwidth = (length(ks)/2)*max_freq/(Fs/2); % half bandwidth, measured as # of bins below max_freq
% for LPF centered at 0, to avoid transcribing higher notes
minbin = floor(length(ks)/2 - bandwidth);
maxbin = floor(length(ks)/2 + bandwidth);

notes = strings(1,num_gabors);
freqs = zeros(1, num_gabors);
for j = 1:num_gabors
    [M,I] = max(Sgt_spec(minbin:maxbin,j)); I = I + minbin;
    % [M,I] = max(Sgt_spec(:,j));
    if M > threshold
        freqs(j) = abs(ks(I));
    else
        freqs(j) = 0;
    end
end

subplot(2,1,2);

stairs((1:length(freqs))*(song_portion*tr)/length(freqs)+45, freq2midi(freqs));
```

```
xlabel("Time (s)");
ylabel("MIDI note");
title("MIDI score")
set(gca, 'Fontsize', fontsize);
%% trash this too??
track = y(1:floor(end*song_portion));
sz = length(track);
tau = linspace(0, tr*song_portion, num_gabors);  % gabor centers, unit seconds

t = (1:sz)/Fs; % song sample time-values
t = cast(t, 'single');
Sgt_spec = cast(zeros(floor(sz/frequency_step),num_gabors), 'single');

gauss = @(t, tau) exp(-a*(t-tau).^2);
% Perform gabor transform, store in Sgt_spec
for j = 1:num_gabors
    g = gauss(t, tau(j));
    g = cast(reshape(g,sz,1), 'single');
    Sg = g.*track;
    Sgt = fft(Sg);
    Sgt = Sgt(1:frequency_step:end);
    % (un)-comment the line below to resolve off-by-one errors
    Sgt = Sgt(1:end-1);
    Sgt_spec(:,j) = cast(fftshift(log(1+abs(Sgt))), 'single');
end
clear Sgt Sg g;

% Display spectrogram
subplot(2,1,1);
ks = linspace(-Fs/2, Fs/2, sz/frequency_step);
pcolor(tau, ks, Sgt_spec)
shading interp
set(gca, 'ylim', [600, 6000], 'Fontsize', 12)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
title("Spectrogram of Comfortably Numb's guitar solo. Computed with " + num_gabors + " STFTs using a="
title("First " + floor(song_portion*tr) + " seconds with " + num_gabors + " a = " + a + " STFTs");

% Let's get the notes.
bandwidth = (length(ks)/2)*max_freq/(Fs/2); % half bandwidth, measured as # of bins below max_freq
% for LPF centered at 0
minbin = floor(length(ks)/2 - bandwidth);
maxbin = floor(length(ks)/2 + bandwidth);

notes = strings(1,num_gabors);
freqs = zeros(1, num_gabors);
for j = 1:num_gabors
    [M,I] = max(Sgt_spec(minbin:maxbin,j)); I = I + minbin;
    % [M,I] = max(Sgt_spec(:,j));
    if M > threshold
        freqs(j) = abs(ks(I));
    else
        freqs(j) = 0;
    end
end
subplot(2,1,2);
stairs((1:length(freqs))*(song_portion*tr)/length(freqs), freq2midi(freqs));
xlabel("Time (s)");
ylabel("MIDI note");
title("MIDI score")
```

B.2 freq2midi.m

```
function [midi] = freq2midi(freq)
% Input vector of frequencies
%   Not guaranteed to work with array
% Outputs the 7-bit MIDI channel representation of pitch
%   Does so by comparing against C0.
%   From a base frequency, I multiply 2^{1/12} until I get up to the target
%       frequency. Thus compute log_1.0595 (target/base) to get the num of
%       semitones above base. Then compare mod 12 against the map to get
%       the note. Then do /12 to find out how many octaves to increase by.
%       Counting system works best with C0 as the base; otherwise a
%       semitone offset is necessary.
    c0 = 16.35160;
    thresh = 0.5;
    semitones = log(freq./c0)/log(2^(1/12));
    % semitones = cast(floor(semitones+thresh), "uint8");
    semitones = floor(semitones + thresh);

    midi = semitones + 12;
end
```

B.3 freq2note.m

```
function [note] = freq2note(freq)
% Input vector of frequencies
% Outputs 'B3' or whatever the note is for each frequency as row vector
%   Does so by comparing against C0.
%   From a base frequency, I multiply 2^{1/12} until I get up to the target
%       frequency. Thus compute log_1.0595 (target/base) to get the num of
%       semitones above base. Then compare mod 12 against the map to get
%       the note. Then do /12 to find out how many octaves to increase by.
%       Counting system works best with C0 as the base. Otherwise a
%       semitone offset is necessary.
    c0 = 16.35160;
    thresh = 0.5;
    pitchSet = ["C","C#","D","D#","E","F","F#","G","G#","A","A#","B"];

    freq = reshape(freq, 1, length(freq));
    note = strings(1, length(freq));

    for j = 1:length(freq)
        if freq(j) == 0
            note(j) = "R";
        else
            semitones = log(freq(j)/c0)/log(2^(1/12));
            semitones = floor(semitones + thresh);

            pitch = pitchSet(mod(semitones, 12)+1); % 1-based indexing
                                                    % hurts
            octave = string(floor(semitones/12));

            note(j) = strcat(pitch, octave);
        end
    end
end
```

B.4 gnrlowpass48.m

```
function y = gnrlowpass48(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.8 and DSP System Toolbox 9.10.
% Generated on: 10-Feb-2021 15:09:43

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass = 950;    % Passband Frequency
    % Fstop = 1050;   % Stopband Frequency
    % Apass = 1;      % Passband Ripple (dB)
    % Astop = 30;     % Stopband Attenuation (dB)
    % Fs    = 48000;  % Sampling Frequency
    %
    % h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 2 1 1 -1.97466284727406 0.990600286548679; 1 2 1 1 ...
        -1.95632536848088 0.972114806563741; 1 2 1 1 -1.93847509116368 ...
        0.954120460241966; 1 2 1 1 -1.92119185259571 0.936697729220625; 1 2 1 1 ...
        -1.90454821886639 0.919919765267672; 1 2 1 1 -1.8886097484062 ...
        0.903852655933539; 1 2 1 1 -1.87343531883007 0.888555754032459; 1 2 1 1 ...
        -1.85907749933776 0.874082053048491; 1 2 1 1 -1.84558295278325 ...
        0.860478592449971; 1 2 1 1 -1.83299285348195 0.847786878868872; 1 2 1 1 ...
        -1.82134330876833 0.836043311060775; 1 2 1 1 -1.81066577417361 ...
        0.825279598433459; 1 2 1 1 -1.80098745381356 0.815523164666239; 1 2 1 1 ...
        -1.79233167912762 0.806797529505828; 1 2 1 1 -1.7847182604756 ...
        0.799122663200798; 1 2 1 1 -1.77816380727353 0.792515309221253; 1 2 1 1 ...
        -1.77268201333993 0.786989271908051; 1 2 1 1 -1.76828390493967 ...
        0.782555666518701; 1 2 1 1 -1.7649780496723 0.779223129801552; 1 2 1 1 ...
        -1.76277072487294 0.776997989755917; 1 2 1 1 -1.76166604460053 ...
        0.775884393645142], ...
        'ScaleValues', [0.00398435981865421; 0.00394735952071483; ...
        0.00391134226957109; 0.00387646915622848; 0.00384288660032074; ...
        0.0038107268818351; 0.00378010880059779; 0.00375113842768193; ...
        0.00372390991667967; 0.00369850634672938; 0.00367500057311003; ...
        0.00365345606496261; 0.00363392771316935; 0.00361646259455146; ...
        0.0036011006813005; 0.00358787548692993; 0.00357681464203008; ...
        0.0035679403947568; 0.00356127003231401; 0.00355681622074339; ...
        0.00355458726115368; 1]);
end

s = double(x);
y = step(Hd,s);
```

B.5 lowpassbass.m

```
function y = lowpassbass(x)
%DOFILTER Filters input x and returns output y.

% MATLAB Code
% Generated by MATLAB(R) 9.8 and DSP System Toolbox 9.10.
% Generated on: 10-Feb-2021 15:34:00

%#codegen

% To generate C/C++ code from this function use the codegen command.
% Type 'help codegen' for more information.

persistent Hd;

if isempty(Hd)

    % The following code was used to design the filter coefficients:
    %
    % Fpass = 800;    % Passband Frequency
    % Fstop = 1000;   % Stopband Frequency
    % Apass = 1;      % Passband Ripple (dB)
    % Astop = 30;     % Stopband Attenuation (dB)
    % Fs    = 44100;  % Sampling Frequency
    %
    % h = fdesign.lowpass('fp,fst,ap,ast', Fpass, Fstop, Apass, Astop, Fs);
    %
    % Hd = design(h, 'butter', ...
    %     'MatchExactly', 'stopband', ...
    %     'SystemObject', true);

    Hd = dsp.BiquadFilter( ...
        'Structure', 'Direct form II', ...
        'SOSMatrix', [1 2 1 1 -1.96663181911825 0.980605634829726; 1 2 1 1 ...
        -1.92971662697755 0.943428143418313; 1 2 1 1 -1.89559738503733 ...
        0.909066468708293; 1 2 1 1 -1.86493644081554 0.87818766449871; 1 2 1 1 ...
        -1.83825697665118 0.851318630577631; 1 2 1 1 -1.81596059400813 ...
        0.828863821970417; 1 2 1 1 -1.79834577077889 0.811123837391098; 1 2 1 1 ...
        -1.78562501684544 0.798312696701242; 1 2 1 1 -1.77793927990389 ...
        0.790572349094603; 1 1 0 1 -0.887684354967378 0], ...
        'ScaleValues', [0.00349345392786972; 0.00342787911018992; ...
        0.00336727091773962; 0.00331280592079279; 0.00326541348161207; ...
        0.00322580699057269; 0.00319451665305211; 0.00317191996395034; ...
        0.00315826729767798; 0.0561578225163112; 1]);
end

s = double(x);
y = step(Hd,s);
```

B.6 removeDuplicate.m

```
function [output] = removeDuplicate(vec)
% Input vector of notes
% Output will remove duplicates
    output = strings(1,length(vec));
    i = 1;
    offset = 0;
    while i < length(vec)
        output(i-offset) = vec(i);
        if vec(i) == vec(i+1)
            offset = offset + 1;
        end
        i = i + 1;
    end
    output = output(1:end-offset);
end
```