AMATH 482: Rock & Roll and the Gabor Transform

Matthew Mangione

February 7, 2021

Abstract

Given audio files of Sweet Child O' Mine by Guns N' Roses and Comfortably Numb by Pink Floyd, we were tasked with using Fourier Analysis to separate the frequencies of each instrument and to identify their musical score. In order to preserve the notes of the songs in time, we utilize the Gabor Transform, as well as Gaussian filtering in Fourier space, to isolate the frequencies of specific instruments.

1 Introduction and Overview

According to Professors Jason Bramburger and Nathan Kutz, 'Sweet Child O' Mine' by Guns N' Roses and 'Comfortably Numb' by Pink Floyd are two of the greatest rock and roll songs of all time, and thus worthy of our analysis. We were given a 14 second long portion of the introductory guitar riff from Sweet Child O' Mine, and a 60 second segment of the second guitar solo in Comfortably Numb. We were tasked with:

1. Reproducing the music score for the guitar riff in the Guns N' Roses clip and the bass line from the Pink Floyd clip.
2. Filter the frequencies in Comfortably Numb to isolate and recreate the bass line.
3. Filter and isolate the guitar solo in Comfortably Numb.

In order to recreate the music scores from the respective audio clips, we utilize the Gabor transform to preserve the location of each note in the song. This allows us to break apart the instruments and the frequencies of each of the notes played, allowing us to infer the architecture of the song. To reduce the noise of other instruments and their overtones, we filter the audio data in Fourier space for each moment in the Gabor transform to isolate a given range of frequencies. Similarly, in our recreations of Comfortably Numbs' bass line and guitar solo, we filter the audio clip in Fourier Space using the Fourier Transform to recreate audio files of the isolated instruments.

2 Theoretical Background

Critical to decomposing data into individual frequencies, the Fourier transform is at the heart of our understanding. Since our data is not continuous, we require the use of the Discrete Fourier Transform, which approximates the Fourier Series of the inputted discrete data.

$$\hat{x_k} = \frac{1}{N}\sum_{n=0}^{N} x_n(\cos\frac{2\pi kn}{N} + i\sin\frac{2\pi kn}{N}) = \frac{1}{N}\sum_{n=0}^{N} x_n e^{\frac{2\pi ikn}{N}} \tag{1}$$

This transformation is implemented using the Fast Fourier Transform (FFT), an algorithm for computing the Discrete Fourier Transform which performs at a pace of `O(NlogN)` when inputting 2^n points. The speed of this algorithm allows us to compute a related measure, the Gabor Transform. While the Fourier Transform decomposes a signal into its corresponding frequencies, information in Fourier space lacks any location in time. In order to compensate for this, Hungarian physicist Gábor Dénes created a variation of the Fourier

Transform by filtering the inputted times with a function **g**, which localize the transformation around the location of the filter. Given the Fourier Transform of a function **f**:

$$\hat{F}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{2}$$

The Gabor Transform can be found by multiplying by the filter function **g**, often a Gaussian:

$$G[f](t, w) = \int_{-\infty}^{\infty} f(x) g(x - t) e^{-iwx} dx \tag{3}$$

Because of the speed of the Fast Fourier Transform, this transformation can be computed for smaller windows of the data's time frame, across all windows available in the data. This allows us to localize the frequencies captured by the FFT to a distinct moment in time, letting us infer information about the location of specific signals.

3 Algorithm Implementation and Development

3.1 Initialization & Pre-Processing

We receive the data from both songs in `.m4a` format, which can be read as a vector of amplitudes using the `audioread` command in MATLAB. To prepare for using the Fast Fourier Transform, we initialize a vector mapping to spectral space, shifting the indices used by the MATLAB implementation of the Discrete Fourier Transform, and dividing each value to represent the frequencies in Hertz (s^{-1}).

3.2 High/Low Pass Filtering

Once we have discovered key frequencies that encompass the full range of the individual instruments, we create a rectangular filter to isolate the frequencies of a given instrument, silencing noise created by others. This is created by creating a vector that corresponds to every possibly value of frequencies in our domain. The indices of this vector which correspond to the desired range of frequencies are set to 1, while all other indices are set to 0.

3.3 Gabor Transform

We use the Gabor Transform to locate frequencies in time. To perform this transformation which captures information about time and frequency, we loop over each instant in our time domain and take the Fourier Series of each window of time. The windows are formed by multiplying a Gaussian filter around the vector in the time domain centered at the moment in time, and then the `fft` command is performed upon the now localized data. When all iterations of this process are viewed together, the changes in frequencies can be viewed throughout time, letting us pick out the exact moment certain frequencies are featured.

Non-max suppression is then utilized to denoise unwanted frequencies and overtones. This is accomplished by finding the strongest frequency in each iteration of the loop, and centering a Gaussian filter around it. When applied to the Fourier transform of the window, only the strongest frequencies remain, i.e., the `max()`. From this point, the frequencies of the desired instrument can be clearly displayed with a spectrogram of all of the Fourier transformations through time.

4 Computational Results

Starting with the guitar riff in Sweet Child 'O Mine, the audio clip was converted into a vector, and over 0.1 second intervals of the 14 second clip, a Gaussian filter was created at each time and multiplied by the original data. We then applied the Fast Fourier Transform to this product, which stored information about the frequencies in the audio clip at each time frame. The vectors created at each iteration were then saved and plotted in Figure 3. The maximum value from each vector in Fourier space was saved, representing the

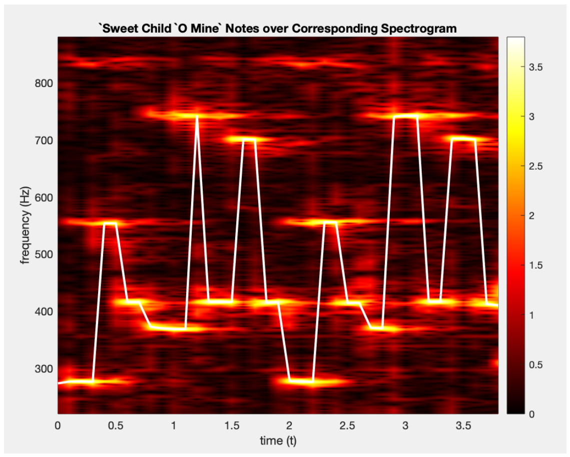

Figure 1: Two measures of the guitar riff overlaid with plot of maximum frequencies.

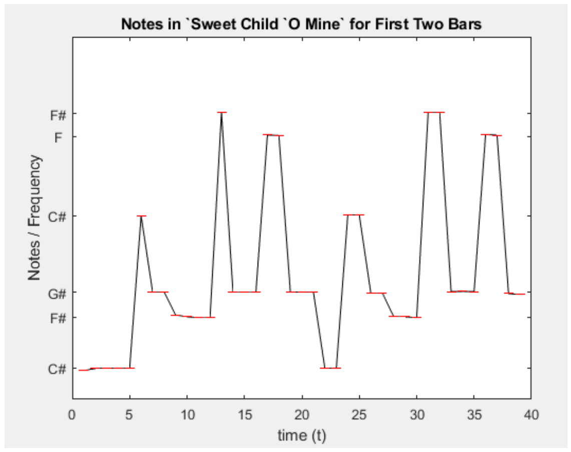

Figure 2: The notes of the first two measures of the guitar riff.

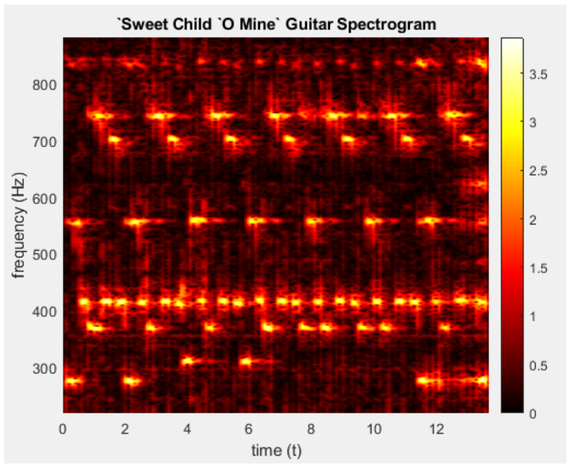

Figure 3: The spectrogram of 'Sweet Child O' Mine' throughout the entire audio clip.

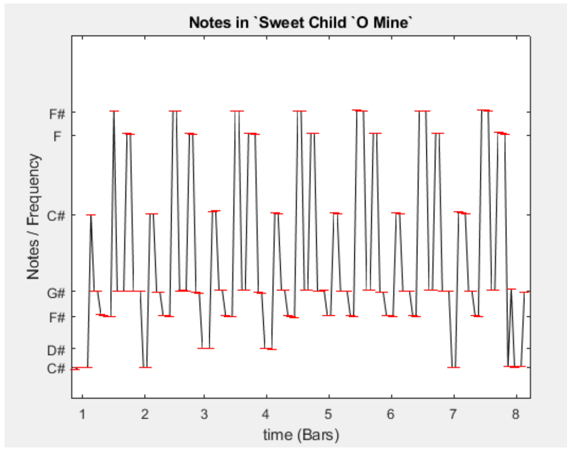

Figure 4: All notes played by the guitar in this clip. Observe the changing tonic note at the beginning of every two measures.

most prominent frequency heard at a point in time. To ensure we only record the frequencies of the guitar, a rectangular filter was later enabled using the discovered range of frequencies played by the guitar. These values were then plotted to represent the melody of the guitar riff, which neatly map up to the actual notes of the song. In Figure 1, the first two measures are plotted on a spectrogram, overlaid their respective notes, shown in Figure 2. This main riff is repeated with differing tonic roots, which follow a progression of (C#, D#, F#, C#). In Figure 4, these differing notes can be found at the beginning of every two measures of the clip, while the rest of the notes in the riff are identical.

We then moved on to the 60 second clip from Comfortably Numb, which included a much more complex musical transcription, featuring the song's second guitar solo, a rhythm guitar, and a bass line that repeats throughout the track. First we isolated and scored the bass line using a process identical to isolating the guitar riff in Sweet Child O' Mine. The frequencies of the bass were neatly contained between 82 Hz and 123 Hz, or notes 'E' to 'B'. When this range was targeted using a rectangular filter, the bass line was shown to repeat 4 times, mainly playing 'B', and descending to the 'E' quickly at the end of each repetition. This progression can be seen in Figures 5 and 6. In order to reproduce the isolated bass line in a new audio file, the rectangular filter for bass frequencies was applied to the Fourier transform of the entire song. When reverted

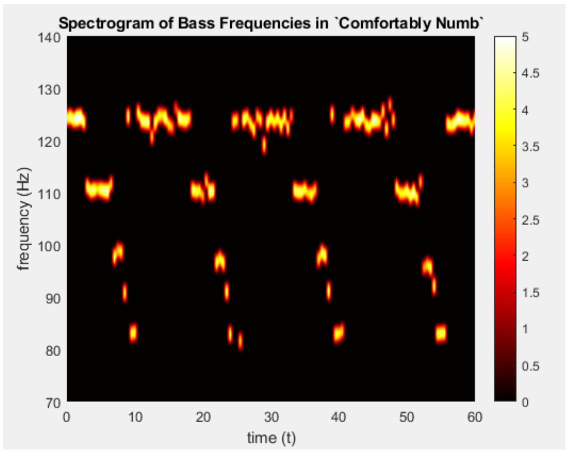

Figure 5: The filtered spectrogram of the 'Comfortably Numb' bass line throughout the entire audio clip.

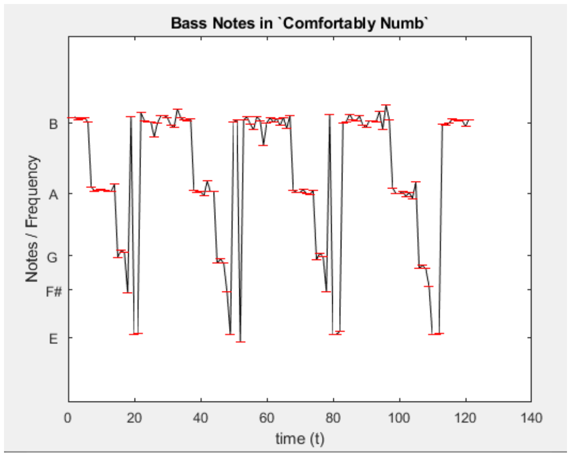

Figure 6: The notes of the bass line throughout the entire 'Comfortably Numb' audio clip.

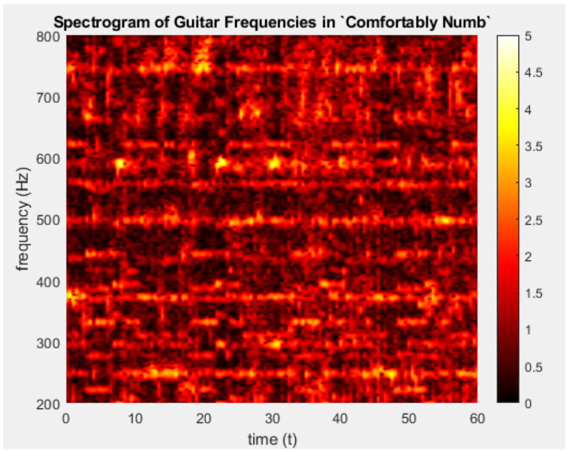

Figure 7: Unfiltered spectrogram of the guitar solo over the entire audio clip.

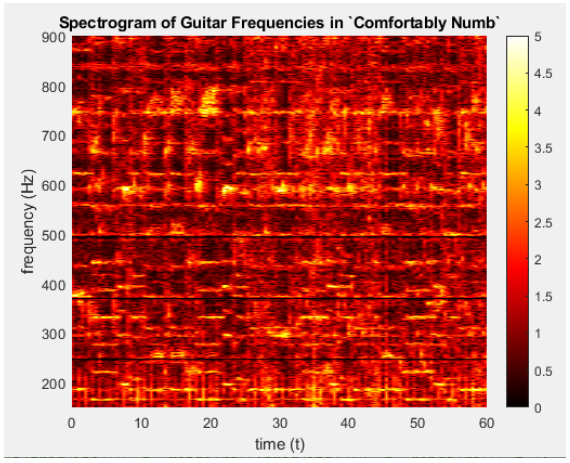

Figure 8: The same spectrogram but with major frequencies of the rhythm guitar muted.

back to the time domain using the `ifft` and `ifftshift` commands, only the bass frequencies remained, which can be founded attached as '`comfortably_numb_bass_iso.m4a`'.

The guitar solo of Comfortably Numb proved more difficult to isolate, as the frequencies of the guitar solo overlap with the rhythm guitar, which plays chords corresponding to the notes of the bass line. Because of this, removing the bass notes with a rectangular filter is not as effective, but is still largely functional. This file without the bass notes can also be found attached to this project as '`comfortably_numb_guitar_solo.m4a`'. This similarly affected the creation of the guitar solo's musical transcript, which included maximum frequencies corresponding to the rhythm guitar. This silenced some notes from the guitar solo, including some of the rhythm's notes instead. In order to reduce this effect, we were forced to make a more precise attempt at filtering the data. For the majority of the clip, the rhythm guitar is playing a B minor chord, most notably consisting of notes B and F#. To counteract these strong notes and their overtones, a very specific Gaussian filter was applied to these notes in Fourier space, which can be seen by the black horizontal bars in Figure 8. This was effective in removing a majority of the rhythm guitar, but also removed crucial notes in the guitar solo which shared the same frequencies with the chord and its overtones.

5 Summary and Conclusions

Over this project, two songs were analyzed using spectral analysis. This allowed us to find the frequencies of each instrument and then isolate them in order to discover which notes were being played and when. This process was straightforward the Guns N' Roses clip, which was largely composed of the guitar notes desired for extraction. For this, we compiled an exact transcription of the guitar riff's notes and displayed the changes the progression underwent. In analyzing Pink Floyd's Comfortably Numb, it was found that the conditions were not as suitable for splitting and transcribing all of the instruments. Due to its relative isolation in the song, the bass line was extracted and a neat music score was created for it. However, the guitar solo was difficult to isolate because of its overlapping frequency ranges and notes with the rhythm guitar. Spectral filter did not improve this issue, as the rhythm notes were more prevalent than many of the notes in the guitar solo.

Overall, the project highlights the difficulties associated with the Fourier Uncertainty Principle. The Fourier transform is limited in the sense that it lacks any information about time; however, the Gabor transform, when iterated across a time frame, helps to encode this location such that the time of a frequency can be known. While accuracy about frequency is lost, this method makes our analysis possible. With it, we have found the melodies to the given songs and have been able to isolate them.

Appendix A MATLAB Functions

- `[Y, Fs] = audioread('filename')` returns a vector Y of the inputted audio file, with sample rate Fs.
- `fft(data)` returns the Discrete Fourier Transform of `data` along one dimension, inverting the indices on both sides of the axis.
- `fftshift(dataf)` returns the elements of `dataf` rearranged to undo the shifted indices of the Fast Fourier Transform algorithm. This is accomplished on every dimension of `dataf`.
- `ifft(dataf)` returns the inverse of Discrete Fourier Transform of `dataf` along one dimension.
- `ifftshift(dataf)` returns the elements of `dataf` rearranged to align with the shifted indices of the Fast Fourier Transform algorithm. This is accomplished on every dimension of `dataf`.
- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[M,I] = max(dat, [], 'all')` returns the maximum value `M` of all elements of `dat`. The index of the maximum value amongst the entirety of `dat` is stored at `I`.

Appendix B MATLAB Code

```
% Matthew Mangione
% AMATH 482: ...
% Rock & Roll & the Gabor Transform

clear all; close all;

% ---------------------------- GNR ------------------------------
[x, sample_rate_gnr] = audioread('GNR.m4a');
time_gnr = length(x)/sample_rate_gnr; % record time in seconds

L = time_gnr;
n = length(x);
t2 = linspace(0,L,n+1);
t = t2(1:n); clear t2;
ks = fftshift((1/L)*[0:n/2-1 -n/2:-1]); % wavenumber -> Hertz

a = 128;
b = 2;
tau = 0:0.1:L;

guitar_rect_filt = zeros(1, n);
guitar_rect_filt(331937:342345) = 1; % positive indices (plots
guitar_rect_filt(316773:327000) = 1; % negative indices (works)

Sgf_spec(:,:) = zeros(length(ks), length(tau));
Sgf_spec(:,:) = zeros(length(ks), length(tau));

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2);
    % Window function
    Sgf = abs(fftshift(fft(g.*(x')))); % fft of data * gaussian time filter
    Sgf = Sgf .* guitar_rect_filt;
    [M, I] = max(Sgf);
    peaks(j) = -ks(I);
    Sgf_spec(:,j) = Sgf;%.*gfilt;
end

figure(1) % only first 2 bars
plot_ind = 1:(length(tau));
pcolor(tau(plot_ind), ks, log(Sgf_spec(:, plot_ind) + 1)); hold on;
shading interp
set(gca,'ylim',[220 880])
colormap(hot)
colorbar
xlabel('time (t)');
ylabel('frequency (Hz)');
title('`Sweet Child `O Mine` Guitar Spectrogram');

figure(2)
plot(plot_ind, peaks(plot_ind), 'k', 'LineWidth', 0.01); hold on;
plot(plot_ind, peaks(plot_ind), 'r_', 'MarkerSize', 6);
set(gca,'ylim',[220 880],'Fontsize', 10)
yticks([277.2, 311.13,369.994, 415.3, 554.3653, 698.456, 739.99]);
yticklabels({'C#', 'D#', 'F#', 'G#', 'C#', 'F ', 'F#'})
xticks([3.5, 22.5, 40.5, 59.5, 78.5, 97.5, 116.5, 135.5]);
xticklabels({'1', '2', '3', '4', '5', '6', '7', '8'})
xlabel('time (Bars)');
ylabel('Notes / Frequency');
title('Notes in `Sweet Child `O Mine`');
```

```matlab
% zoom in on only 2 bars of guitar
plot_ind = 1:(length(tau)/3.5);

figure(3) % only first 2 bars
pcolor(tau(plot_ind), ks, log(Sgf_spec(:, plot_ind) + 1)); hold on;
plot(tau(plot_ind), peaks(plot_ind), 'LineWidth', 2, 'Color', 'w');
shading interp
set(gca,'ylim',[220 880])
colormap(hot)
colorbar
xlabel('time (t)');
ylabel('frequency (Hz)');
title('`Sweet Child `O Mine` Notes over Corresponding Spectrogram');

figure(4)
plot(plot_ind, peaks(plot_ind), 'k'); hold on;
plot(plot_ind, peaks(plot_ind), 'r_', 'LineWidth', .5, 'MarkerSize', 6);
set(gca,'ylim',[220 880],'Fontsize', 10)
% yticks([261.626, 277.2, 293.665, 311.127, 329.63, 349.23, 369.994, 391.995, ...
%        415.305, 440, 466.164, 493.89, 523.25, 554.3653, 587.3295, 622.254, 659.255, ...
%        698.456, 739.99, 783.99, 830.61, 880]);
% yticklabels({'C ', 'C#', 'D ', 'D#', 'E ', 'F ', 'F#', 'G ', 'G#', 'A ', 'A#', 'B ', 'C ',...
%             'C#',  'D ', 'D#', 'E ', 'F ', 'F#', 'G ', 'G#', 'A '})
yticks([277.2, 369.994, 415.3, 554.3653, 698.456, 739.99]);
yticklabels({'C#', 'F#', 'G#', 'C#', 'F ', 'F#'})
xlabel('time (t)');
ylabel('Notes / Frequency');
title('Notes in `Sweet Child `O Mine` for First Two Bars');


% ---------------------------- Pink Floyd ------------------------------
clear Sgf g Sgf_spec peaks

[y, sample_rate_pf] = audioread('Floyd.m4a');
time_floyd = length(y)/sample_rate_pf; % record time in seconds

% limit use of song data to reduce RAM
time_start = 0;
time_end = 60;
time_step = .5;


L = time_floyd;
n = length(y);
t2 = linspace(time_start,time_end,n+1);
t = t2(1:n); clear t2;
ks = fftshift((1/L)*[0:n/2-1 -n/2:-1]); % wavenumber -> Hertz

a = 256;
b = 1;
c = .005;
tau = time_start:time_step:time_end;

% define rectangular filter around bass frequencies ([50, 240])
bass_rect_filt = zeros(1, n);
bass_rect_filt(freq2ind(-240, n, range(ks)):freq2ind(-50, n, range(ks))) = 1;
bass_rect_filt(freq2ind(50, n, range(ks)):freq2ind(240, n, range(ks))) = 1;
```

```
% define rectangular filter around guitar frequencies ([230, max])
guitar_rect_filt = ones(1, n-1);
guitar_rect_filt(freq2ind(-240, n, range(ks)):freq2ind(240, n, range(ks))) = 0;

% % isolate guitar solo & write new audio file
solo = fftshift(fft(y'));
g_filt = blockfilter([246.9 329.6], ks, c);
solo = ifft(ifftshift(guitar_rect_filt .* solo(1:(n-1)))); %.* g_filt));
audiowrite('comfortably_numb_guitar_solo.m4a', real(solo), sample_rate_pf);

%isolate bass & write new audio file
bass = fftshift(fft(y'));
bass = ifft(ifftshift(bass_rect_filt .* bass));
audiowrite('comfortably_numb_bass_iso.m4a', real(bass), sample_rate_pf);

% redefine bass rectangular filter to exclude rhythm guitar for the purpose
% of recreating musical score of the bass notes. Lower range to [50, 140]
bass_rect_filt = zeros(1, n);
bass_rect_filt(freq2ind(-140, n, range(ks)):freq2ind(-50, n, range(ks))) = 1;
bass_rect_filt(freq2ind(50, n, range(ks)):freq2ind(140, n, range(ks))) = 1;

% apply Gabor transform to each timeframe of the song using gaussian
% with width a. Apply non-max supression to reduce noise / emphasize notes.
Sgf_spec(:,:) = zeros(length(ks), length(tau));
Sgf_spec_guitar(:,:) = zeros(length(ks), length(tau));

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % localize data in time
    Sgf = abs(fftshift(fft(g.*(y')))); % fft of data * gaussian time filter
    g_filt = blockfilter([246.9, 329.6, 369.9, 493.8, 587.3, 739.9], ks, c);
    %g_filt = blockfilter([], ks, c);

    % isolate bass & apply non-max supression
    Sgf = Sgf.*bass_rect_filt; % apply filter to focus on bass line
    [M, I] = max(Sgf);
    peaks_bass(j) = -ks(I);
    g_filt = exp(-b*((ks+ks(I)).^2));
    Sgf_spec(:,j) = Sgf(1:length(ks)).*g_filt;

    % isolate guitar solo & apply non-max supression
    Sgf = Sgf(1:(n-1)).*guitar_rect_filt; % apply filter to focus on bass line
    Sgf = Sgf(1:(n-1)) .* g_filt;
    [M, I] = max(Sgf);
    peaks_guitar(j) = -ks(I);
    g_filt = exp(-b*((ks+ks(I)).^2));
    Sgf_spec_guitar(:,j) = Sgf .*g_filt;
end

figure(5) % spectrogram of bass notes
plot_ind = 1:length(tau);
pcolor(tau(plot_ind), ks, log(Sgf_spec(:, plot_ind) + 1)); hold on;
%plot(tau(plot_ind), peaks(plot_ind), 'LineWidth', 2);
shading interp
set(gca,'ylim',[70 140])
caxis([0 5])
colormap(hot)
colorbar
xlabel('time (t)');
ylabel('frequency (Hz)');
title('Spectrogram of Bass Frequencies in `Comfortably Numb`');
```

```
figure(6) % approximate score of bass line
plot(plot_ind, peaks_bass(plot_ind), 'k'); hold on;
plot(plot_ind, peaks_bass(plot_ind), 'r_', 'LineWidth', .5, 'MarkerSize', 6);
set(gca,'ylim',[70 140])
yticks([82.41, 91.49, 97.99, 110.0, 123.4]);
yticklabels({'E ', 'F#', 'G ', 'A ', 'B '})
xlabel('time (t)');
ylabel('Notes / Frequency');
title('Bass Notes in `Comfortably Numb`');

figure(7) % guitar solo spectrogram
plot_ind = 1:length(tau);
pcolor(tau(plot_ind), ks, log(Sgf_spec_guitar(:, plot_ind) + 1)); hold on;
%plot(tau(plot_ind), peaks(plot_ind), 'LineWidth', 2);
shading interp
set(gca,'ylim',[200 800])
caxis([0 5])
colormap(hot)
colorbar
xlabel('time (t)');
ylabel('frequency (Hz)');
title('Spectrogram of Guitar Frequencies in `Comfortably Numb`');


figure(8)
plot(plot_ind, peaks_guitar(plot_ind), 'k'); hold on;
plot(plot_ind, peaks_guitar(plot_ind), 'r_', 'LineWidth', .5, 'MarkerSize', 6);
set(gca,'ylim',[250 600],'Fontsize', 10)

%yticks([82.41, 91.49, 97.99, 110.0, 123.4]);
%yticklabels({'E ', 'F#', 'G ', 'A ', 'B '})
xlabel('time (t)');
ylabel('Notes / Frequency');
title('Guitar Notes in `Comfortably Numb`');

% approximates the index of a given frequency in a vector
function [ind] = freq2ind(freq, n, range)
    slope = n / range;
    zed = n/2;
    ind = round(slope * freq + zed);
end

% creates a filter that blocks all frequencies of a given vector v
function [g_filt] = blockfilter(v, ks, c)
    g_filt = ones(1, length(ks));
    for i = 1:length(v)
        g_filt = g_filt - exp(-c*((ks-ks(freq2ind(v(i), length(ks)+1, range(ks)))).^2));
        g_filt = g_filt - exp(-c*((ks-ks(freq2ind(-v(i), length(ks)+1, range(ks)))).^2));
    end

end
```

Listing 1: Corresponding MATLAB code for this project.