Rock & Roll and the Gabor Transform

Yukai Yan

February 10th, 2021

Abstract

In this paper, we try to analyze a portion of two of the greatest rock and roll songs of all time, that is, the clips of the songs *Sweet Child O' Mine* by Guns N' Roses(GNR) and *Comfortably Numb* by Pink Floyd(Floyd). This paper aims to reproduce the music score of the guitar in the GNR clip and the bass in the Floyd clip. We can achieve this goal by utilizing the Gabor transform. Later in the paper, we will also try to identify the guitar solo after filtering the bass in the Floyd clip.

1 Introduction and Overview

1.1 The Rock & Roll problem

Sweet Child O' Mine by Guns N' Roses(GNR) and *Comfortably Numb* by Pink Floyd(Floyd) are famous in Rock & Roll. *Guitar World* ranked the GNR riff # 37 all-time and the Floyd riff #4 all-time. Given a 14-second clip of GNR, we try to reproduce the music score of the guitar in this clip. The data is obtained in an m4a file. To import and convert them into vectors representing the music, we use *audioread* command in MatLab. The clip of Floyd is also recorded in a 60-second m4a file. Using a similar strategy, we will be able to identify the bass in this clip.

1.2 Approach the problem

Unfortunately, looking at the vectors representing the music is not ideal for reproducing the music score. An experienced musician can achieve the goal by listening to the music, but there might be some deviations. This paper uses frequency to accurately identify the music score, which can be achieved by the Gabor transform, or the short-time Fourier transform (STFT). The Fourier transform takes a function of time, x, and converts it to a function of frequencies, k. However, the shift-invariance of the absolute value of the Fourier transform loses information about what is happening in the time domain. The Gabor transform, named after Dennis Gabor, is then used to solve this issue. It is capable of determining the sinusoidal frequency and phase content of local sections of a signal as it changes over time. After filtering, we will use the spectrum to convey the frequency information as the window slides over the domain.

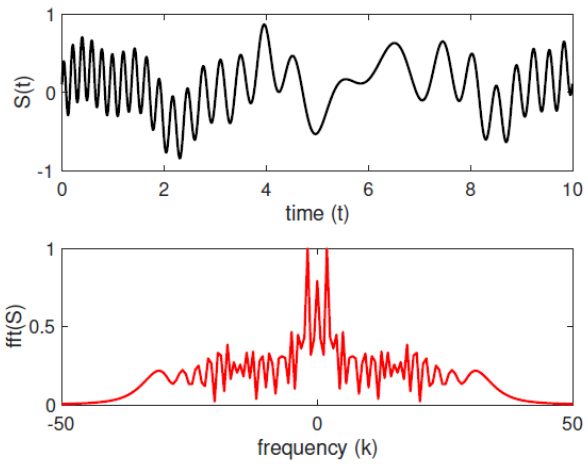

Figure 1: The Fourier transform converts a function of time to a function of frequencies. However, we lose information on the time-domain. We cannot identify when the high frequencies present.

1.3 Overview

We will discuss Fourier Transform, Gabor transform, filtering, and spectrogram in Section Two. In Section Three, we will develop the solution and implement the corresponding algorithms. Section Four will include the computational results derived from the approach, as the summary and conclusion will be in Section Five.

2 Theoretical Background

2.1 Fourier Transform

Suppose we are given a function $f(x)$ with $x \in \mathbb{R}$, we can define the Fourier Transform of $f(x)$, written $\hat{f}(k)$ by:

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

We can also recover $f(x)$ using the inverse Fourier Transform:

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

By the Euler's formula, we know that e^{ikx} can be defined by $sin(kx)$ and $cos(kx)$, where the value of k gives the frequency of sine and cosine waves. Therefore, the Fourier transform takes a function of space or time, x, and converts it to a function of frequencies, k.

2.2 Gabor Transform

Let us consider a filter function $g(t)$. Then, shifting by τ and multiplying by a function $f(t)$ represents the filtered function $f(t)g(t-\tau)$, with the filter centred at τ. *The Gabor transform*, or STFT, is then given precisely by

$$\tilde{f}(\tau, k) = \int_{-\infty}^{\infty} f(t) g(t-\tau) e^{-ikt} dt$$

That is, for a fixed τ the function $\tilde{f}(\tau, k)$ gives us information about the frequency components near time τ
The inverse of the Gabor transform is given by

$$f(t) = \frac{1}{2\pi ||g||_2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \tilde{f}(\tau, k) g(t-\tau) e^{ikt} dk d\tau$$

In this paper, the Gaussian filter will be our default choice for $g(t)$ as Gabor developed this method using Gaussian as well. The Gaussian filter is defined by:

$$F(k) = e^{-a(k-\tau)^2}$$

Where a determines the width of the filter and τ determines the centre of the filter.

2.3 Spectrogram

A Spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. It helps us convey the information as we stack all the Fourier transforms next to each other to make a plot with the horizontal direction being the value of τ (window center) and the vertical direction being the frequency. With Spectrogram, we can identify the frequency in a specific time range.

2.4 Filters

In order to get the music score of the guitar in the Floyd clip, we need to filter out the bass. In this project, a band-pass filter helps us identify the data of interest. The filter will pass frequencies within a specific range and rejects frequencies outside that range. We will use it to filter data in the frequency space before the Gabor transform.

3 Algorithm Implementation and Development

Since our goal is to identify the music score in the clip, it is essential for us to record the time information in the frequency space. To achieve this, we will use the Gabor transform. The code can be into four main parts:

1. Load data and identify the spatial/frequency domains.
2. (Optional) Identify the data of interest by the range of frequencies.
3. Apply the Gabor transform.
4. Identify the music score by plotting the spectrum

Note that the code implemented in MatLab is included in Appendix B.

Algorithm 1: Load Data and identify the spatial/frequency domains

Clean work space
Load data from two m4a files using *audioread*
Define time t based on the sample rate for that data.
Set the spatial domain L to be the end of t, and the Fourier modes n to be the length of t
Set the frequency domain k with $k = \frac{1}{L}[0 : (\frac{n}{2} - 1) \quad -\frac{n}{2} : -1]$
Set ks as the fftshift of k

Note that MatLab scales out the 2π in the periods of oscillation to make the periods integers, that is, $e^{i2\pi ft}$ for frequency $|f|$, while we are using the notation e^{ikt} for frequency $|k|/2\pi$. f is measured in Hertz and k is called angular frequency. Therefore, we need to rescale k in the spectrograms by $1/L$ instead of $2\pi/L$

Algorithm 2: (Optional) Identify the data of interest by the range of frequencies

Define the filter to be the data of interest in the frequency space.
Take the Fourier transform of the audio data.
Multiply that data by the filter.
Take the Inverse Fourier transform of that data

The filter can be any filters you like.In this paper, however, we will use the filter mentioned earlier in the Section 2.4. Note that we took the fast Fourier transform in this step, and MatLab requires fftshift to obtain the correct transformation.

Algorithm 3: Apply the Gabor transform

Define the step size *diff* of windows
Define τ to be the values between 0 and L with step size *diff*
Set the width of window $a = 100$
Initialize an empty vector v to store data
for $j = 1 : length(\tau)$ **do**
    Define the Gaussian filter with a and $\tau(j)$
    Multiply the audio data by the filter.
    Take the Fourier transform of that data.
    Divide data by the maximum.
    Record the data in the vector v.
end

With the vector v, we can plot the spectrogram in the next step.

Algorithm 4: Identify the music score by plotting the spectrum

plot τ, ks, and v using *pcolor*
Set the y-axis to the desired range.
Set the colorMap as desired.

The results of the computed data (along with the graph) are included in Section 5.

4 Computational Results

Before we start, we can plot the data to get a general idea of how it looks like:

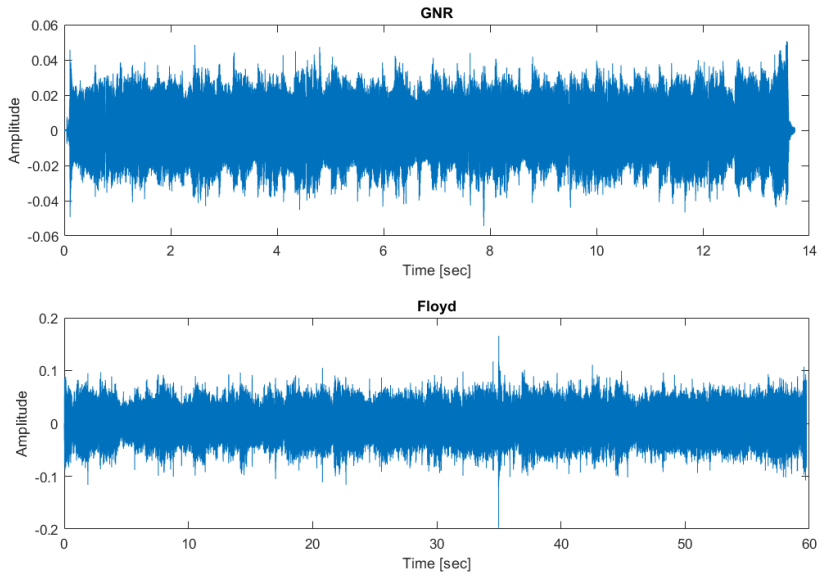

Figure 2: The converted vectors representing the audio data

The plots tell us nothing about the music score. Hence, we will use the Gabor transform to find frequencies along with the time information. For reference, we can also plot the transformation process in progress:

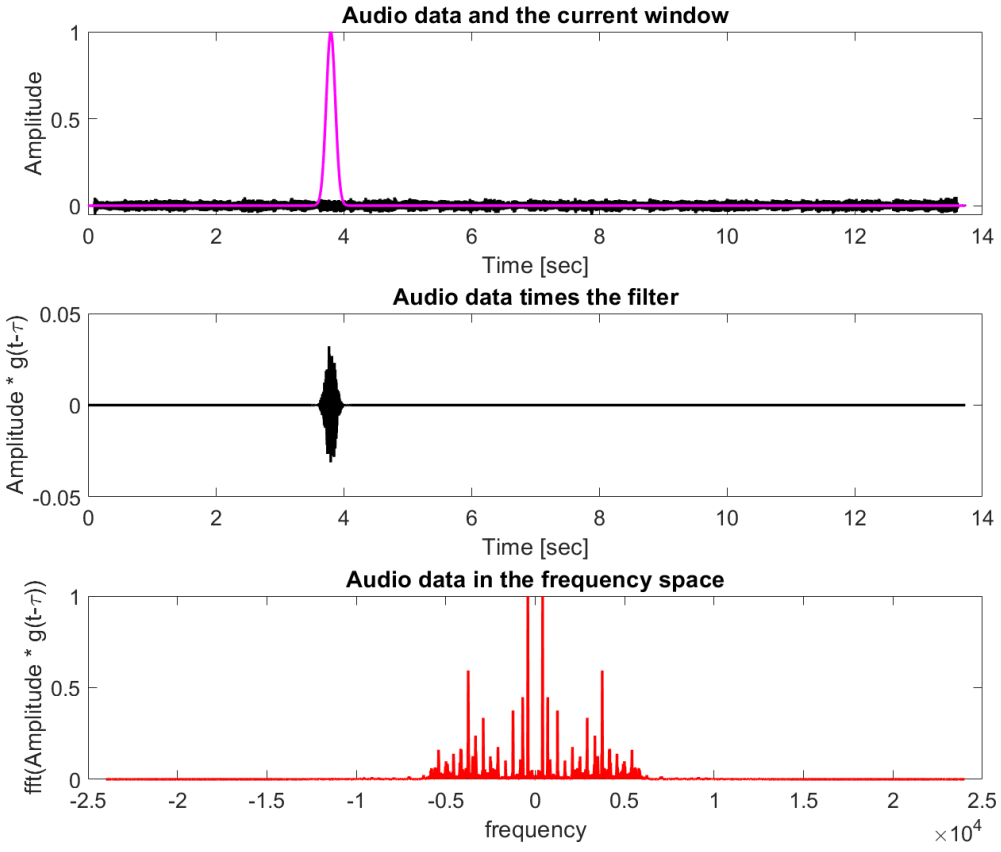

Figure 3: The Gabor transform in process for *Sweet Child O' Mine*

We can now use the information recorded in the frequency space to plot the spectrogram.

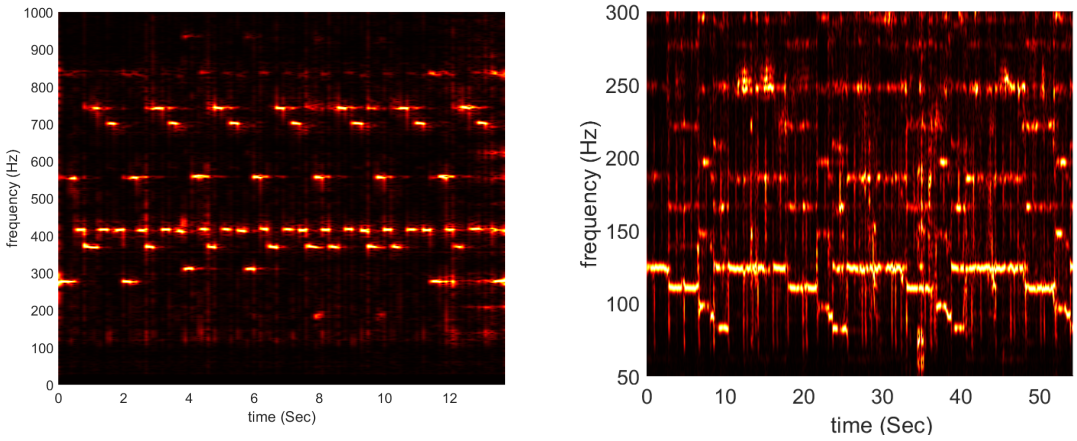

Figure 4: The Spectrogram for Guitar in GNR and Bass in Floyd, respectively

As you can see in the Spectrogram for GNR, the music score for the guitar is clearly identified by the bright spots, while fewer bright spots indicate overtones. Note that it is common for the guitar to play two strings of the same music score together to produce harmonious sound, we will combine the same music score at a given time t. Using the given music chart, we can find the music score for the main melody: **D, A, G, A, F, A**, and these beats repeat themselves roughly every two seconds.

Similarly, we can do the same for Floyd. Note that it is not clear as the previous graph, this is due to the mix of a variety of instruments used in that clip. Luckily, we can still identify the music score of bass by the bright spots, that is, **B, A, G, F, E**, and these beats repeat themselves roughly every fifteen seconds.

As you might already know, the bass typically operates in the range of 60 to 250 Hz, and we can use a band-pass filter mentioned earlier in section 2.4 to help us isolate bass in the frequency space. Then, we can use the similar strategy to plot the spectrogram:

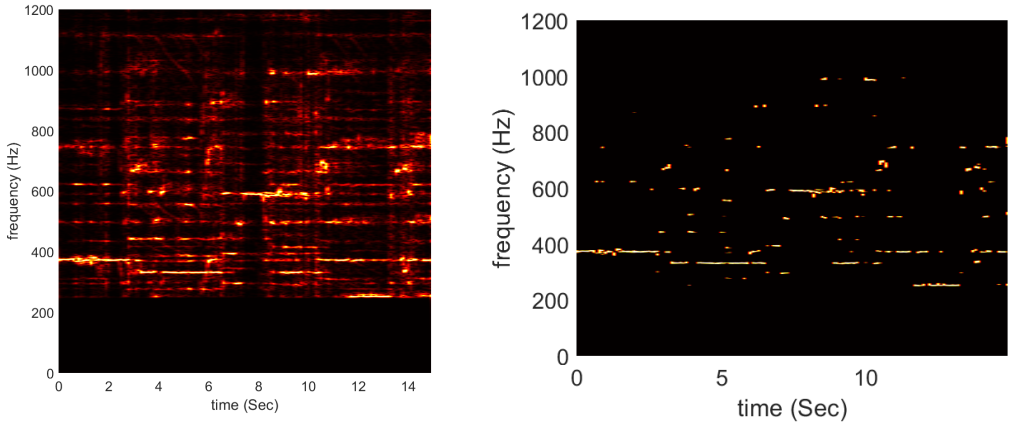

Figure 5: Spectrogram for guitar in Floyd, and the spectrogram with .7 cutoff, respectively

Note that we only plot the spectrogram for the first fifteen seconds, as the melody repeats itself in that time range. A limited range like this can help us significantly improve the run-time complexity. As you can see in the graph, unfortunately, it is almost impossible to extract the music score of the guitar. The idea of pulling out frequency by its maximum is not feasible here anymore. With a .7 cutoff (everything below 0.7 are treated as 0), we can hardly see the music score of guitar.

5 Summary and Conclusions

The Fourier transform allows us to transform a function of time into a function of frequencies. However, it loses information about what is happening in the time domain. In this paper, the Fourier transform is not enough to identify the music score at a specific period. The Gabor transform is then used to analyze the music clips. As demonstrated in this paper, The Gabor transform, or windowed Fourier transform, can identify the music score by producing the Spectrogram. The primary instruments that dominated the music are clearly identifiable. However, due to the richness of the music and the overtone of the devices, it might be troublesome to identify the music score of accompaniments. More work can be done in the future to design a specific filter and cleans noise and unwanted data.

Appendices

Appendix A: MatLab functions used and brief explanations

The following list includes the MatLab functions used in Appendix B. It also contains a brief implementation explanation of their functionalities. More information can be found at MatLab Documentation at MathWorks.

- *[y, Fs] = audioread(filename)*: reads data from the file named *filename*, and returns sampled data, y, and a sample rate for that data, Fs.
- L = *length(X)* returns the length of the largest array dimension in X
- y = *fftshift(X)*: rearrange a Fourier transform X by shifting the zero-frequency components to the center of the array.
- Y = *fftn(X)*: returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm.
- X = *ifftn(Y)*: returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm.
- Y = *exp(X)*: returns the exponential e^x for each element in array X.
- Y = *abs(X)* returns the absolute value of each element in array X.
- M = *max(A)* returns the maximum elements of an array.
- *pcolor(C)* creates a pseudocolor plot using the values in matrix C
- *shading interp* varies the color in each line segment and face by interpolating the colormap index or true color value across the line or face
- *colormap map* sets the *colormap* for the current figure to one of the predefined colormaps.

Appendix B: MatLab Code

```
%% load and plot data
clear variables; close all; clc;

showgraph = true; % plot the 'animation' in for loop

figure(1)
[y_gnr, Fs1] = audioread('GNR.m4a');
t_gnr = (1:length(y_gnr))/Fs1; % record time in seconds
subplot(2,1,1)
plot(t_gnr,y_gnr);
xlabel('Time [sec]'); ylabel('Amplitude');
title('GNR');

[y_floy, Fs2] = audioread('Floyd.m4a');
t_floy = (1:length(y_floy))/Fs2;
subplot(2,1,2)
plot(t_floy,y_floy);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Floyd');
%p8 = audioplayer(y,Fs); playblocking(p8);

%% GNR Gabor Transform
L = t_gnr(end);    % spatial domain
n = length(t_gnr); % Fourier modes
k = (1/L)*[0:(n/2 - 1) -n/2:-1]; ks = fftshift(k); % even

figure(2)
diff = 0.1;       % step size
tau = 0:diff:L;   % centre of window
a = 100;          % window size;
gnr_spec = [];
for j = 1:length(tau)
    g = exp(-a *(t_gnr - tau(j)).^ 2); %Gaussian
    yf_floy = g'.*y_gnr;
    yft_gnr = fft(yf_floy);

    % clean data for better spectrogram
    yft_spec = fftshift(abs(yft_gnr))/max(abs(yft_gnr));
    gnr_spec(:,j) = yft_spec;

    % plot the graph
    if (showgraph)
        subplot(3,1,1)
        plot(t_gnr, y_gnr, 'k', t_gnr, g, 'm', 'Linewidth',2);
        set(gca,'Fontsize',16)
        xlabel('Time [sec]'), ylabel('Amplitude')
        title('Audio data and the current window');
        drawnow

        subplot(3,1,2),
        plot(t_gnr,yf_floy, 'k', 'Linewidth',2)
        set(gca,'Fontsize',16,'ylim',[-0.05, 0.05])
        xlabel('Time [sec]'), ylabel('Amplitude * g(t-\tau)');
        title('Audio data times the filter');
        drawnow

        subplot(3,1,3),
        plot(ks,yft_spec, 'r', 'Linewidth',2)
```

```
59          set(gca,'Fontsize',16)
60          xlabel('frequency'); ylabel('fft(Amplitude * g(t-\tau))');
61          title('Audio data in the frequency space');
62          drawnow
63
64          pause(0.1)
65      end
66
67  end
68
69  %% plot the spectrogram for GNR
70  figure(3)
71  pcolor(tau,ks,log(gnr_spec + 1))
72  shading interp
73  set(gca,'ylim',[0,1000],'Fontsize',16)
74  colormap(hot)
75  xlabel('time (Sec)'), ylabel('frequency (Hz)')
76
77
78  %% Floyd Gabor Transform for bass
79  % optional cutoff(first half)
80  % only evaluate part of data due to the limitation of computing power
81  % cutoff = (length(t_floy)+1)/4 + 1;
82  % t_floy = t_floy(1:cutoff);
83  % y_floy = y_floy(1:cutoff);
84
85  % setup
86  L = t_floy(end);    % spatial domain
87  n = length(t_floy); % Fourier modes
88  k = (1/L)*[0:(n/2) (-n/2):-1]; ks = fftshift(k); %odd
89
90  % data of interest to reduce run-time complexity
91  filter = abs(ks) <= 250 & abs(ks) >= 50;  % frequency range for bass
92  yft_floy = fftshift(fft(y_floy)).*filter';
93  yf_floy = ifft(fftshift(yft_floy));
94
95  figure(2)
96  diff = 0.1;       % step size
97  tau = 0:diff:L;   % centre of window
98  a = 100;          % window size;
99  floy_spec = [];
100 for j = 1:length(tau)
101     g = exp(-a *(t_floy - tau(j)).^ 2); %Gaussian
102     yf_floy = g .* y_floy';
103     yft_floy = fft(yf_floy);
104
105     % clean data for better spectrogram
106     yft_spec = fftshift(abs(yft_floy))/max(abs(yft_floy));
107     floy_spec(:,j) = yft_spec;
108
109     % plot the graph
110     if (showgraph)
111         subplot(3,1,1)
112         plot(t_floy, y_floy, 'k', t_floy, g, 'm', 'Linewidth',2);
113         set(gca,'Fontsize',16),
114         xlabel('Time [sec]'), ylabel('Amplitude')
115         drawnow
116
117         subplot(3,1,2),
118         plot(t_floy,yf_floy, 'k', 'Linewidth',2)
119         set(gca,'Fontsize',16,'ylim',[-0.1, 0.1])
```

```
        xlabel('Time [sec]'), ylabel('Amplitude * g(t-\tau)');
        drawnow

        subplot(3,1,3),
        plot(ks,yft_spec, 'r', 'Linewidth',2)
        set(gca,'Fontsize',16)
        xlabel('frequency'); ylabel('fft(Amplitude * g(t-\tau))');
        drawnow

        pause(0.1)
    end

end

%% plot the spectrogram for Floy
figure(4)
% spectrogram(y_floy)
pcolor(tau,ks,floy_spec)
shading interp
set(gca,'ylim',[50,300],'Fontsize',16)
colormap(hot)
xlabel('time (Sec)'), ylabel('frequency (Hz)')

%% Apply filters to clean bass in Floy
% May need to run the first section again before running this part
% optional cutoff(first 15 sec)
cutoff = (length(t_floy)+1)/4 + 1;
t_floy = t_floy(1:cutoff);
y_floy = y_floy(1:cutoff);

L = t_floy(end);    % spatial domain
n = length(t_floy); % Fourier modes
k = (1/L)*[0:(n/2) (-n/2):-1]; ks = fftshift(k); %odd

% bandPass Filter
% yf_floy = bandpass(y_floy,[250 1200],Fs2);

% Box Filter
% filter = abs(ks) <= 250 & abs(ks) >= 50;  % verify  bass
filter = abs(ks) > 250;
yft_floy = fftshift(fft(y_floy)).*filter';
yf_floy = ifft(fftshift(yft_floy));

% plot the filtered graph
if (showgraph)
    plot(ks,yft_floy, 'r', 'Linewidth',2)
    set(gca,'Fontsize',16, 'ylim', [0, 500])
    xlabel('frequency(Hz)'); ylabel('fft(Floyd) * filter');
    drawnow
end

figure(5)
diff = 0.1;       % step size
a = 200;          % window size;
tau = 0:diff:L;  % centre of window
floy_spec = [];
for j = 1:length(tau)
    g = exp(-a *(t_floy - tau(j)).^ 2); %Gaussian
    yt_floy = g .* yf_floy';
    yft_floy = fft(yt_floy);

```

```
181      % clean data for better spectrogram
182      yft_spec = fftshift(abs(yft_floy))/max(abs(yft_floy));
183      for k = 1:length(yft_spec)
184          if (yft_spec(k) < 0.7)
185              yft_spec(k) = 0;
186          end
187      end
188
189      floy_spec(:,j) = yft_spec;
190
191      % plot the graph
192      if (showgraph)
193          subplot(3,1,1)
194          plot(t_floy, y_floy, 'k', t_floy, g, 'm', 'Linewidth',2);
195          set(gca,'Fontsize',16),
196          xlabel('Time [sec]'), ylabel('Amplitude')
197          drawnow
198
199          subplot(3,1,2),
200          plot(t_floy,real(yt_floy), 'k', 'Linewidth',2)
201          set(gca,'Fontsize',16, 'ylim',[-0.1, 0.1])
202          xlabel('Time [sec]'), ylabel('Amplitude * g(t-\tau)');
203          drawnow
204
205          subplot(3,1,3),
206          plot(ks,yft_spec, 'r', 'Linewidth',2)
207          set(gca,'Fontsize',16)
208          xlabel('frequency'); ylabel('fft(Amplitude * g(t-\tau))');
209          drawnow
210
211          pause(0.01)
212      end
213
214  end
215
216  %% plot the NEW spectrogram for Floy
217  figure(6)
218  pcolor(tau,ks, floy_spec)
219  shading interp
220  set(gca,'ylim',[0,1200],'Fontsize',16)
221  colormap(hot)
222  xlabel('time (Sec)'), ylabel('frequency (Hz)')
```