Homework 2

Audrey Shingleton

February 7, 2020

Abstract

Given audio recordings, I used the Gabor transformation to produce spectrograms. This allowed me to visualize the signals in both the time and frequency domain. I analyzed various parameters of the Gabor transformation and determined the order of notes for the song "Mary had a little lamb" played by both the piano and recorder.

I. Introduction and Overview

I have been given multiple musical recordings and I want to analyze what frequencies are being played and at what time. The Fourier transform can be used to determine the frequency content of a given song, but it fails to capture time information. Therefore, I will need to use the Gabor transform to capture content in both the time and frequency domain simultaneously.

In this homework, I will first investigate various qualities of the Gabor transform, such as the effect of window width and translation size, by producing spectrograms of Handel's "Messiah". For the second part, I will reproduce the music scores and evaluate the differences of "Mary had a little lamb" recordings played by the piano and recorder.

II. Theoretical Background

The Gabor Transform

The Fourier transform is unable to localize a signal in both the time and frequency domains. To solve this, a modification of the Fourier transformation kernel was introduced as defined by:

$$g_{t,w}(\tau) = e^{iw\tau} g(\tau - t) \tag{1}$$

where the new term to the Fourier kernel $g(\tau - t)$ acts as a time filter for localizing the signal around $t = \tau$.[1] The Gabor transform is thus defined as:

$$G[f](t, w) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) e^{-iw\tau} d\tau \tag{2}$$

where various types of Gabor windows $g(\tau - t)$ with window width a, such as a Gaussian or Mexican hat window, can be used. This definition assumes g to be real and symmetric.

In practice, the Gabor transform is computed by discretizing the time and frequency domain.[1] This means that the transform is done on a lattice defined by:

$$v = mw_0 \ \tau = nt_0$$

where m an n are integers and $w_0, t_0 > 0$ are constant. The discrete version of $g_{t,w]}$ is defined by:

$$g_{m,n}(t) = e^{i2\pi m w_0 t} g(t - nt_0) \tag{3}$$

and the Gabor transform becomes

$$\int_{-\infty}^{\infty} f(t) g_{m,n}(t) dt \tag{4}$$

If $0 < t_0, w_0 < 1$ then the signal is oversampled meaning that time frames exists and will produce good localization of the signal in time and frequency. If $w_0, t_0 > 1$ then the signal is under-sampled and the Gabor lattice will not be able to reproduce the signal.[1]

Gabor Transform Drawbacks

There are drawbacks to the Gabor transform. For one, the time window filters out behavior of the signal in a window centered at τ with width a. This means that any portion of the signal with a wavelength longer than the window is lost. Additionally, the time-frequency spread around a Gabor window, computed by the variance, is governed by the Heinsberg uncertainty principle.[1] It trades some amount of accuracy in both the time and frequency domains in order to capture their resolution simultaneously. In other words, the shorter the time filtering window, the less information of the frequency content and vice-versa.

Spectrograms

Spectrograms represent the signal in both the time and frequency domain. They are produced by varying the position of the Gabor time filter and producing spectra at each location in time. As the time filter position is discretized, the level of discretion is important for good time-frequency analysis.

III. Algorithm Implementation and Development

Part 1

For part 1, I was given an audio signal for part of Handel's "Messiah". I determined the length of the audio signal and the number of time points. Then I re-scaled the frequency domain by a factor of $2\pi/L$, where L is the audio length, to perform FFT calculations later on as the Fourier transform assumes a 2π periodic signal.

To calculate the Gabor transform of the signal, I used a Gaussian time filter defined as

$$g(t) = e^{-a(t-tslide)^2)} \tag{5}$$

where a is the filter width and tslide is the length of the audio in 0.1 second increments. For each time slice in tslide, I multiplied the audio signal by the filter to get a windowed section of the signal. I then took the FFT of the section to get local frequency content in time. Finally, I plotted each slice in the time and frequency domain to produce a spectrogram.

To explore the effects of oversampling and undersampling, I modified and compared spectrograms using a small and large translation parameter corresponding to translation step values of 0.01 and 1 seconds. To explore the effects of window width, I produced and compared spectrograms with window widths of 80, 50, 20, 10, 5, 1, 0.5, and 0.1.

Part 2

For part 2, I first prepared the audio signals of "Mary had a little lamb" for a piano and recorder by defining the audio lengths, time points, window width ($a = 100$), tslide (length of audio in 0.1 second increments), etc. Then, using the method for producing a spectrogram as explained in part 1, I produced spectrograms for both recordings.

In order to produce the music scores of the songs, I had to remove the overtones due to the timbre of the instrument. I did this by finding the index of the maximum value for each time slice after taking the FFT of the windowed signal, where the maximum indicates the strongest frequency at that time point. The center frequency of that slice is the frequency value of the FFT shifted frequencies at the maximum index. Multiplying this frequency by 2π gave me the frequency in Hertz to produce the music scores.

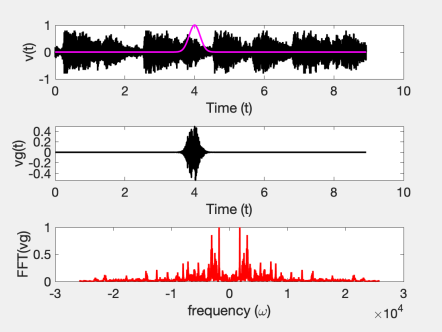

Figure 1: A visualization of the Gabor transform on the audio signal with a Gaussian time filter (window width $a = 20$).

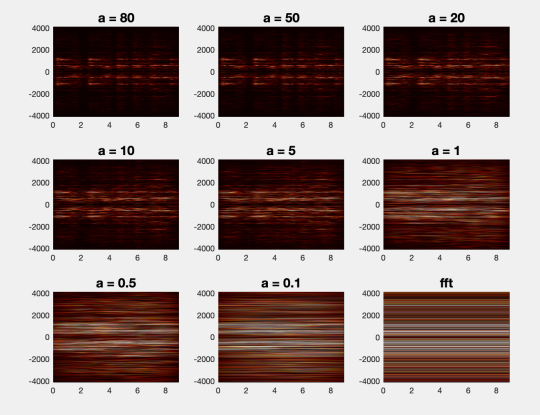

Figure 2: Spectrograms of the signal with varied window widths.

IV. Computational Results

Part 1

I produced a visualization of the Gabor transform, with a translation step value of 0.1, on the audio signal (Figure 1). This allowed me to easily see how the Gabor transform takes windowed sections of the signal to get the frequency content of that time slice. I then produced spectrograms with varying window widths (Figure 2). I saw that using a very narrow window ($a = 80$) gives much less frequency resolution as compared to some of the other widths. I also noticed that there was almost no spectrogram difference between widths $a = 80$ and $a = 20$ despite the large difference in value whereas there was a significant difference between $a = 5$ and $a = 1$. I also found that using a very wide window

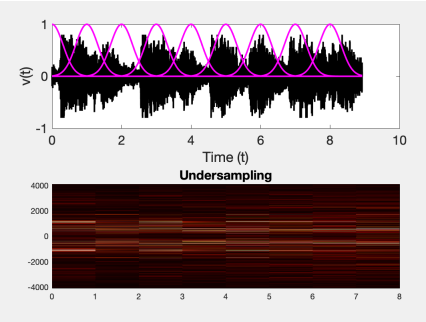

Figure 3: Visualization and spectrogram of undersampled signal.

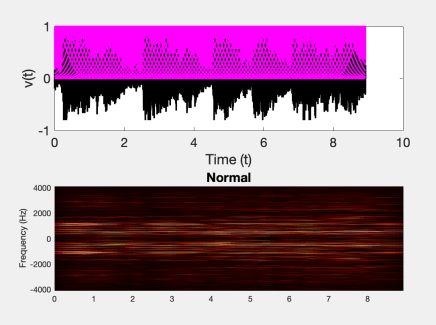

Figure 4: Visualization and spectrogram of moderately sampled signal.

width ($a = 0.1$) gives barely any time information as it is nearly the same as the FFT, which only has frequency content. I decided that a window width of about $a = 10$ was a good balance between capturing a fair amount of frequency and time resolution for this signal.

Using a window width of $a = 10$, I looked at under, moderate, and over sampled signals (Figures 3 through 5). These corresponded to taking the spectra of the time slices of the signal in increments of 1, 0.1, and 0.01 seconds. The over and moderate sampled spectrograms look nearly identical, but the resolution of the under sampled spectrogram looks incomplete.

Part 2

The spectrograms of "Mary had a little lamb" on the piano and recorder show a similar pattern in the order of higher and lower frequencies, but have very different pitches (Figures 6 and 7). I also noticed that the piano seems to produce more overtones and undertones than the recorder. The music scores,

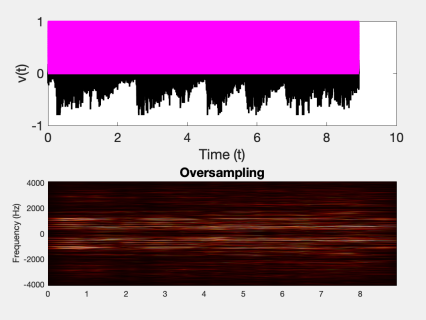

Figure 5: Visualization and spectrogram of oversampled signal.

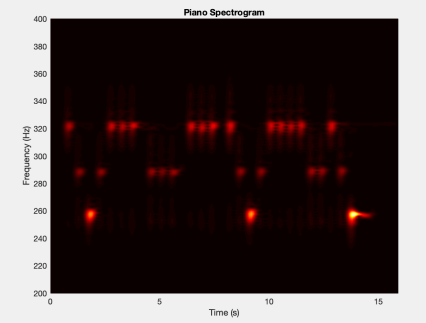

Figure 6: Spectrogram of "Mary had a little lamb" on the piano.

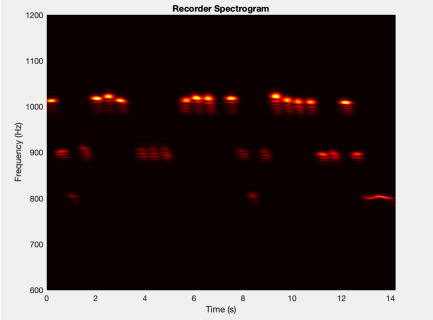

Figure 7: Spectrogram of "Mary had a little lamb" on the recorder.

produced by filtering out the overtones, confirm the fact that the signals have a similar pattern of notes despite the difference in pitch (Figures 8 and 9).

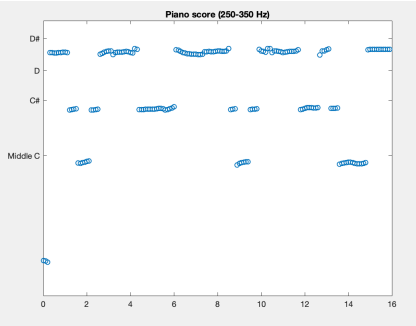

Figure 8: Music score of "Mary had a little lamb" on the piano.

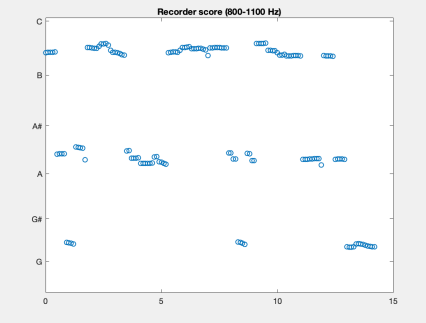

Figure 9: Music score of "Mary had a little lamb" on the recorder.

V. Summary and Conclusions

Part 1

In part 1, I produced and compared spectrograms with various window widths and window translations. As shown in Figure 3, a larger window width has better frequency resolution and vice-versa. It is necessary to evaluate various window widths to find a good balance between frequency and time resolution. I was also curious about how the translation size would effect the spectrograms (Figures 3 through 5). The content of the under-sampled spectrogram was noticeably disconnected. However, it was interesting to find that the moderately sampled and over-sampled spectrograms looked nearly identical. This shows that good results can be produced with a moderate translation window but it is necessary not to under sample in order to capture the signal content well.

Part 2

In part 2, given audio files of "Mary had a little lamb" on the piano and recorder, I was able to produce spectrograms of each piece clearly showing which frequencies were played throughout the recording (Figures 6 and 7). The spectrograms highlight the fact that the recorder plays with much higher frequencies and produces less overtones than the piano. By filtering the overtones, I produced a music score for each piece which show the pieces have a similar note pattern (Figures 8 and 9).

Appendix A. MATLAB functions used and explanations

- fft: Transforms the function to the frequency domain using the Fast Fourier Transformation algorithm
- fftshift: Shifts the function back to its mathematically correct positions for plotting
- max: Returns the maximum value M in a given 3D matrix and its linear index I
- audioread: Opens a wav file as an array of signal data

Appendix B. MATLAB codes

```
% Pt 1
%% Plot signal and FFT
clear; close all; clc;
load handel
v = y';
L = length(v)/Fs; % length of audio
t = (1:length(v))/Fs;
n = length(t);
k = (2*pi/L)*[0:(n-1)/2 -(n-1)/2:-1];
ks = fftshift(k);
vt = fft(v);

figure(1)
subplot(2,1,1)
plot(t,v);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Signal of Interest, v(n)');
```

```
% Fourier domain
subplot(2,1,2)
plot(ks, abs(vt)/max(abs(vt)));
xlabel('frequency (\omega)'), ylabel('FFT(v)')

%% plays music
% p8 = audioplayer(v,Fs);
% playblocking(p8);

%% Construct Gabor window and add to time domain plot
tau = 4; % some time
a = 20; % larger a is narrower
g = exp(-a*(t-tau).^2);
subplot(2,1,1)
plot(t,v,'k',t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

%% Apply filter and take fft
vg = g.*v;
vgt = fft(vg);

figure(2)
subplot(3,1,1)
plot(t,v,'k','Linewidth',2)
hold on
plot(t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

subplot(3,1,2)
plot(t,vg,'k','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('vg(t)')

subplot(3,1,3)
plot(ks,abs(fftshift(vgt))/max(abs(vgt)),'r','Linewidth',2);
set(gca,'Fontsize',16)
xlabel('frequency (\omega)'), ylabel('FFT(vg)')
%% Change window size - narrow window
tau = 4;
a = 80;
g = exp(-a*(t-tau).^2);
vg = g.*v;
vgt = fft(vg);

figure(3)
subplot(3,1,1)
plot(t,v,'k','Linewidth',2)
```

```
hold on
plot(t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

subplot(3,1,2)
plot(t,vg,'k','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('vg(t)')

subplot(3,1,3)
aa = abs(fftshift(vgt))/max(abs(vgt));
plot(ks,aa,'r','Linewidth',2);
set(gca,'Fontsize',16)
xlabel('frequency (\omega)'), ylabel('FFT(vg)')
%% Change window size - wide window
tau = 4;
a = 1;
g = exp(-a*(t-tau).^2);
vg = g.*v;
vgt = fft(vg);

figure(4)
subplot(3,1,1)
plot(t,v,'k','Linewidth',2)
hold on
plot(t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

subplot(3,1,2)
plot(t,vg,'k','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('vg(t)')

subplot(3,1,3)
aa = abs(fftshift(vgt))/max(abs(vgt));
plot(ks,aa,'r','Linewidth',2);
set(gca,'Fontsize',16)
xlabel('frequency (\omega)'), ylabel('FFT(vg)')
%% Calculate Gabor transform and plot spectrogram
a = 50;
tslide=0:0.1:L;
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)); % We don't want to scale it
end
```

```
figure(5)
pcolor(tslide,ks./(2*pi),vgt_spec.'),
shading interp
colormap(hot)
ylabel("Frequency (Hz)");
title('Spectrogram, a = 20')

%% Spectrograms for varying window sizes
% 0.01 looks like FFT, not time information
% 25 better time info, less frequency resolution Heinsberg uncertainity
figure(6)
a_vec = [80 50 20 10 5 1 0.5 0.1];
for jj = 1:length(a_vec)
    a = a_vec(jj);
    tslide=0:0.1:L;
    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(3,3,jj)
    pcolor(tslide,ks./(2*pi),vgt_spec.'),
    shading interp
    title(['a = ',num2str(a)],'Fontsize',16)
    colormap(hot)
end
vgt_spec = repmat(fftshift(abs(vt)),length(tslide),1);
subplot(3,3,9)
pcolor(tslide,ks./(2*pi),vgt_spec.'),
shading interp
title('fft','Fontsize',16)
colormap(hot)

%% oversampling
a = 5;

figure(7)
subplot(2,1,1)
plot(t,v,'k','Linewidth',2)
hold on
tslide=0:0.01:L;
for j=1:length(tslide)
    g = exp(-a*(t-tslide(j)).^2);
```

```
    subplot(2,1,1)
    plot(t,g,'m','Linewidth',2)
end
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

subplot(2,1,2)
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

pcolor(tslide,ks./(2*pi),vgt_spec.'),
shading interp
title('Oversampling','Fontsize',16)
colormap(hot)
ylabel("Frequency (Hz)");

%% normal sampling
a = 5;

figure(8)
subplot(2,1,1)
plot(t,v,'k','Linewidth',2)
hold on
tslide=0:0.1:L;
for j=1:length(tslide)
    g = exp(-a*(t-tslide(j)).^2);
    subplot(2,1,1)
    plot(t,g,'m','Linewidth',2)
end
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

subplot(2,1,2)
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

pcolor(tslide,ks./(2*pi),vgt_spec.'),
shading interp
```

```
title('Normal','Fontsize',16)
colormap(hot)
ylabel("Frequency (Hz)");

%% undersampling
a = 5;

figure(9)
subplot(2,1,1)
plot(t,v,'k','Linewidth',2)
hold on
tslide=0:1:L;
for j=1:length(tslide)
    g = exp(-a*(t-tslide(j)).^2);
    subplot(2,1,1)
    plot(t,g,'m','Linewidth',2)
end
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

subplot(2,1,2)
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

pcolor(tslide,ks./(2*pi),vgt_spec.'),
shading interp
title('Undersampling','Fontsize',16)
colormap(hot)

%% Piano
[y,Fs] = audioread('music1.wav');

L = length(y)/Fs; % record time in seconds
t = (1:length(y))/Fs;
n = length(y);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);
tslide_p=0:0.1:L;

v = y'; % transpose of audio
vt = abs(fft(v));
vt_norm = fftshift(vt/max(vt));
```

```
figure(1)
subplot(2,1,1)
plot(t,y);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (piano)');
%p8 = audioplayer(y,Fs); playblocking(p8);

% Fourier domain
subplot(2,1,2)
plot(ks, vt_norm);
xlabel('frequency (\omega)'), ylabel('FFT(v)')

tau = 10; % some time
a = 25; % larger a is narrower
g = exp(-a*(t-tau).^2);
subplot(2,1,1)
plot(t,v,'k',t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

%% Piano spectrogram

piano_notes = [];
vgt_spec = zeros(length(tslide_p),n);

% figure(3)
for j=1:length(tslide_p)
    g = exp(-a*(t-tslide_p(j)).^2);
    vg = g.*v;
    vgt=abs(fft(vg));
    [M, I] = max(vgt);
    piano_notes = [piano_notes; abs(k(I))/(2*pi)];
    vgt_spec(j,:) = fftshift(vgt);
%     subplot(2,1,1), plot(t, vg, 'k')
%     subplot(2,1,2), plot(ks,fftshift(vgt/max(vgt)))
%     drawnow
%     pause(0.1)
end

%% Plot spectrogram
% min(piano_notes) around 200
% max(piano notes) around 350
figure(4)
pcolor(tslide_p,ks./(2*pi),vgt_spec.'), shading interp
xlabel("Time (s)"); ylabel("Frequency (Hz)"); title("Piano Spectrogram");
set(gca,'Ylim',[200 400])
```

```
colormap(hot)

%% Recorder
[y,Fs] = audioread('music2.wav');

L = length(y)/Fs; % record time in seconds
t = (1:length(y))/Fs;
n = length(y);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);
tslide_r=0:0.1:L;

v = y'; % transpose of audio
vt = abs(fft(v));
vt_norm = fftshift(vt/max(vt));

figure(1)
subplot(2,1,1)
plot(t,y);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
% p8 = audioplayer(y,Fs); playblocking(p8);

% Fourier domain
subplot(2,1,2)
plot(ks, vt_norm);
xlabel('frequency (\omega)'), ylabel('FFT(v)')

tau = 10; % some time
a = 25; % larger a is narrower
g = exp(-a*(t-tau).^2);
subplot(2,1,1)
plot(t,v,'k',t,g,'m','Linewidth',2)
set(gca,'Fontsize',16), xlabel('Time (t)'), ylabel('v(t)')

%% Recorder spectrogram

recorder_notes = [];
vgt_spec = zeros(length(tslide_r),n);

% figure(3)
for j=1:length(tslide_r)
    g = exp(-a*(t-tslide_r(j)).^2);
    vg = g.*v;
    vgt=abs(fft(vg));
    [M, I] = max(vgt);
```

```
    recorder_notes = [recorder_notes; abs(k(I))/(2*pi)];
    vgt_spec(j,:) = fftshift(vgt);
%     subplot(2,1,1), plot(t, vg, 'k')
%     subplot(2,1,2), plot(ks,fftshift(vgt/max(vgt)))
%     drawnow
%     pause(0.1)
end
%% Plot recorder spectrogram
figure(5)
pcolor(tslide_r,ks./(2*pi),vgt_spec.'), shading interp
xlabel("Time (s)"); ylabel("Frequency (Hz)"); title("Recorder Spectrogram");
set(gca,'Ylim',[600 1200])
colormap(hot)

%% Plot notes
figure(6)
plot(tslide_p, piano_notes, 'o');
yticks([261.63,293.66,311.13,329.63,349.23, 369.99]);
yticklabels({'Middle C','C#','D','D#','E','F', 'F#'});
title("Piano score (250-350 Hz)");

figure(7)
plot(tslide_r, recorder_notes, 'o');
yticks([783.99,830.61,880,932.33,987.77,1046.5,1108.7]);
yticklabels({'G','G#','A','A#','B','C','C#'});
title("Recorder score (800-1100 Hz)");
```

References

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