ECE280 - Lab 1: Music Synthesis

Jerry Worthy (jdw102)

February 11, 2022

I have adhered to the Duke Community Standard in completing this assignment.

Contents

1 Introduction

- The Beatles are widely considered to be the most influential band of all time, and because of their dynamic mix of genres, interesting musical techniques, and creative songwriting, they remain my favorite band to this day. The song I chose was She Said She Said from The Beatles' 1966 album, Revolver.This album continued to build off the sharp change in direction that was begun in their previous work Rubber Soul. It represented a rapid shift from The Beatles' status as a new age pop band to a more experimental, studio band, and I believe this song reflects this pivotal shift in sound to a tee. It is comprised of three very distinct parts that I thought would be perfect to pick apart during the course of this lab. This includes the melody of John Lennon singing, the backing of the rhythm guitar chords, and the lead guitar's solos and riffs. I felt that the guitar in this song had a very unique sound to it, which would be fun to try and emulate using MATLAB and that the creation of the three different components would allow a lot of room for exploring the different transformations of the sound functions.

- Effects Used

  1. ADSR
  2. Harmonics
  3. Frequency and Volume Modulation

2 Procedure

- The notes were generated using variations of a function I called play. This function would take in the volume, frequency, and duration of the note as a parameter, and use cosine to produce an array that could be converted to sound using MATLAB's soundsc function. The volume would act as the amplitude, while the specified note determined the frequency. The notes ranged along two octaves, spanning 110 hZ to 440 hZ. The duration of the notes, ranging from eighth to whole notes, were evenly spaced arrays of that increased by the inverse of the fundamental frequency (8000 hZ) until the end of the appropriate time. This produced an array the size of the corresponding duration values that held all the appropriate cosine values for each point in the duration array. Three separate arrays were created for each of the three different parts of the song, the melody, the chords, and the lead guitar. To maintain consistent size across the three arrays, the same note lengths were used at each point in the arrays, which meant that larger notes that overlapped with smaller notes in different parts had to be broken up across multiple notes and rests were used when one part was not playing.There were eight different variations of the play function. Four of these variations were used for the lead guitar and chords section, as they all had the same effects applied to them to mimic the sound of the guitar on the track. The other four were specifically for the melody and mimicked the sound of a piano to replace the vocals on the track. For each set of four variations, there was one function that played an individual note, one that played the beginning of a held note, one that played the middle of a held note, and one that played the end of a held note. This allowed a continuous note to be broken up into smaller notes across the different arrays. There were three different chords used, Aflat, Bflat, and Eflat. These chords were made by adding the three notes that comprise them and assigning it to a variable. Each chord had 8 different variations, corresponding to the four different note lengths and whether it was played once or supposed to be held.

- Explanation of Effects

  1. ADSR

     The ADSR effect was used to make the notes sound more like they were being produced by an instrument rather than the computer. This was done using a piece-wise function that first raises the volume gradually, holds it, then decreases it back to close to silence. This piece-wise function was stored in a variable then multiplied element-by-element with the note produced by the play function.

2. Harmonics
   The harmonic effect was done using the reference that provided the fourier transformations of a note based on an instrument. Essentially, the cosines of different integer multiples of the note's frequency were added back to the base note's cosine value, with each different harmonic's amplitude being multiplied by a different value. There were two different harmonic variations in my program, with the guitar and melody play functions each having their own unique harmonic compositions up until the eleventh harmonic of a note.
3. Frequency and Volume Modulation
   The guitar in the track I chose has a very sharp and unique sound. I tried to emulate this by modulating both the frequency and volume of the play function for the guitar. This was done by multiplying a decaying exponential by the volume of the base note and by the frequencies of both the 2nd and 3rd harmonic, since these had the greatest weights in the harmonic composition.

3 Results

The final sound was definitely recognizable as the chosen track, with the guitar sound being relatively similar. There are points in my song where the backing track disappears to give more prominence to the melody, and this does not occur in the original track. The notes overlap fairly well between the songs, as does the tempo and general structure, with the riffs and solos lining up nicely. However, there are very noticeable differences in the tone of the guitar sounds which likely could be addressed given more time. Also, the use of the piano sound as the melody compared to the voice of John Lennon does make quite the difference in the overall atmosphere of the piece.

4 Discussion

- Discuss how each effect individually affects the sound.

  A) Multiple Notes: playing multiple notes at the same time allows for the creation of chords, which produce a richer and more versatile sound that can complement melodies.

  B) Exponential Decay: this mimics the actual decay of the volume of the note and can be used to vary the frequency of the note, creating a quieter note that flattens slightly as it plays.

  C) Harmonics: this causes the note to sound more rich and musical, as it mimics the harmonics that are naturally produced by instruments.

  D) Instrument Synthesis: mimicking instruments can make the vector sound more like real music with actual instruments, such as the emulation of a guitar in my track.

  E) Reverb/Echo: this can mimic the effect of the sound that is produced by the acoustics of a room when an instrument is played, making it sound more realistic.

- What challenges did you encounter, if any, when implementing each effect? How did you resolve the challenges?
  When implementing ADSR, it was hard to keep the sound consistent when using notes that were meant to be held across multiple lengths of notes. To counteract this I created specific play functions for held notes as previously described and edited the ADSR function to make it sound consistent. When implementing the harmonics it was difficult to decide how many harmonics to use, but I opted to keep adding harmonics until the Fourier-series chart was approximately zero. When implementing the frequency and volume modulation, it was especially hard to decide which harmonic to implement which modulation, as it could result in completely changing the sound to something less to be desired. Ultimately I relied on trial and error to decide how to properly implement this effect by testing each different harmonic and modulation.

- Which effects had the greatest impact on the sound? Which had the least impact? Why?
  The effect that had the greatest impact was the harmonic. This really elevated the quality of the sound

being produced by each note by making it sound much more realistic and pleasing to the ear. I believe this is because this was the biggest jump to how instruments actually produce notes. The effect that had the least great effect on the sound was the frequency modulation. This was because using the decaying exponential function made its modulation very slight, which a very subtle ambience to the sound of the emulated guitar, but in comparison to the other effects was not that noticeable.

- Assuming you had more time, how would you make the song better? Why do you think this would improve your song?
  Firstly, I would want to add a reverb effect specifically to the chords section of the song, as I believe this would make a better compliment to the melody and also sound much more like the original track. I would also want to experiment more with the frequency modulation to try and mimic the guitar better. This would include trying other functions and choosing different harmonics to modulate.

- Comments on specific questions.

  I. Why is the endpoint $1 - \frac{1}{f_s}$ instead of 1?
  Because the duration of each note is evenly spaced by the inverse of the fundamental frequency, depending on how long the duration is, this may not fit perfectly within the length of the vector. To account for this we subtract one inverse from the total length so that it is guaranteed to fit.

  II. Does it matter if you use sin or cos?
  No, both functions will work. They will produce the same vector shifted by a quarter of the total period, which will almost always be indistinguishable to the ear.

  III. Why do you need to use vectors of zero amplitude?
  This is used for rests in the song, and was especially useful when there was one part playing while another part was not.

  IV. What happens when the amplitude signal exceeds the range between -1 and 1?
  This increases the volume of the note produced by the signal.

5 MATLAB^{TM} code

```
%initialize sampling frequency
fs = 8000;

%create array of note frequencies
note = zeros(1, 24);
for i = 0:23
    note(i+1) = 110*(2^(i/12));
end
%creating lower octave notes
AL = note(1);
AsharpL = note(2);
BL = note(3);
CL = note(4);
CsharpL = note(5);
DL = note(6);
DsharpL = note(7);
EL = note(8);
FL = note(9);
FsharpL = note(10);
GL = note(11);
GsharpL = note(12);
Rest = 0;
```

```
%creating higher octave notes
A = note(13);
Asharp = note(14);
B = note(15);
C = note(16);
Csharp = note(17);
D = note(18);
Dsharp = note(19);
E = note(20);
F = note(21);
Fsharp = note(22);
G = note(23);
Gsharp = note(24);

%create time vectors
eigth = [0:1/fs:0.2625-2.1/fs];
quarter = [0:1/fs:0.525-2.1/fs]; %[start:increment:end]
half = [0:1/fs:1.05-2.1/fs];
whole = [0:1/fs:2.1-2.1/fs];

cvol = 0.5;
%creating chord values
Bflate = play(cvol, Asharp, eigth) + play(cvol, D, eigth) + play(cvol, F, eigth)
Bflatq = play(cvol, Asharp, quarter) + play(cvol, D, quarter) + play(cvol, F, quarter)
Bflath = play(cvol, Asharp, half) + play(cvol, D, half) + play(cvol, F, half)
Bflatw = play(cvol, Asharp, whole) + play(cvol, D, whole) + play(cvol, F, whole)

Bflates = susplay(cvol, Asharp, eigth) + susplay(cvol, D, eigth) + susplay(cvol, F, eigth)
Bflatqs = susplay(cvol, Asharp, quarter) + susplay(cvol, D, quarter) + susplay(cvol, F, qu
Bflaths = susplay(cvol, Asharp, half) + susplay(cvol, D, half) + susplay(cvol, F, half)
Bflatws = susplay(cvol, Asharp, whole) + susplay(cvol, D, whole) + susplay(cvol, F, whole)

Aflate = play(cvol, Gsharp, eigth) + play(cvol, C, eigth) + play(cvol, Dsharp, eigth)
Aflatq = play(cvol, Gsharp, quarter) + play(cvol, C, quarter) + play(cvol, Dsharp, quarter)
Aflath = play(cvol, Gsharp, half) + play(cvol, C, half) + play(cvol, Dsharp, half)
Aflatw = play(cvol, Gsharp, whole) + play(cvol, C, whole) + play(cvol, Dsharp, whole)

Aflates = susplay(cvol, Gsharp, eigth) + susplay(cvol, C, eigth) + susplay(cvol, Dsharp, ei
Aflatqs = susplay(cvol, Gsharp, quarter) + susplay(cvol, C, quarter) + susplay(cvol, Dsharp
Aflaths = susplay(cvol, Gsharp, half) + susplay(cvol, C, half) + susplay(cvol, Dsharp, hal
Aflatws = susplay(cvol, Gsharp, whole) + susplay(cvol, C, whole) + susplay(cvol, Dsharp, w

Eflate = play(cvol, Dsharp, eigth) + play(cvol, G, eigth) + play(cvol, Asharp, eigth)
Eflatq = play(cvol, Dsharp, quarter) + play(cvol, G, quarter) + play(cvol, Asharp, quarter)
Eflath = play(cvol, Dsharp, half) + play(cvol, G, half) + play(cvol, Asharp, half)
Eflatw = play(cvol, Dsharp, whole) + play(cvol, G, whole) + play(cvol, Asharp, whole)

Eflates = susplay(cvol, Dsharp, eigth) + susplay(cvol, G, eigth) + susplay(cvol, Asharp, ei
Eflatqs = susplay(cvol, Dsharp, quarter) + susplay(cvol, G, quarter) + susplay(cvol, Asharp
Eflaths = susplay(cvol, Dsharp, half) + susplay(cvol, G, half) + susplay(cvol, Asharp, hal
Eflatws = susplay(cvol, Dsharp, whole) + susplay(cvol, G, whole) + susplay(cvol, Asharp, w
```

```
%lead guitar part
lead = [   susplayb(1, Asharp, eigth), susplay(1, C, eigth), susplay(1, D, eigth), susplay
           susplaye(1, Asharp, quarter), Bflatq, Bflate, Bflate, Bflatq,...
           play(1, Asharp, quarter), Bflatq, Bflate, Bflate, Bflatq,...
           play(0, Rest, quarter), play(0, Rest, quarter), play(0, Rest, quarter), play(0
           play(0, Rest, quarter), play(0, Rest, quarter), play(0, Rest, eigth),play(0, R
           play(0, Rest, quarter), play(0, Rest, eigth), play(0, Rest, eigth), play(1, Ash
           play(1, D, quarter), play(1, Asharp, quarter), play(0, Rest, eigth), play(0, R
           play(0, Rest, quarter), play(0, Rest, eigth), play(0, Rest, eigth), play(1, Ash
           play(1, D, eigth), play(1, Dsharp, eigth), play(1, F, eigth), susplayb(1, Ashar
           play(0, Rest, eigth), play(0, Rest, eigth), play(0, Rest, eigth), play(0, Rest
           play(0, Rest, eigth), play(0, Rest, eigth), play(0, Rest, eigth), play(0, Rest
           play(1, D, eigth), play(1, C, eigth), play(1, Asharp, eigth), susplayb(1, C, ei
           play(1, GsharpL, eigth), play(1, FL, eigth), play(1, C, eigth), susplayb(1, Ash
           ...
           playm(0, Rest, quarter), susplaymb(0, Rest, quarter), susplaym(0, Rest, quarter
           susplayme(0, Rest, quarter), playm(0, Rest, quarter), playm(0, Rest, eigth), pl
           playm(0, Rest, quarter), susplaymb(0, Rest, eigth), susplaym(0, Rest, eigth), s
           playm(0, Rest, quarter), playm(0, Rest, quarter), playm(0, Rest, eigth), playm(
           playm(0, Rest, quarter), playm(0, Rest, eigth), playm(0, Rest, eigth), susplaym
           play(0, Rest, whole)
               ]
%create melody part
melody = [ playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, R
           playm(0, Rest, quarter), playm(0, Rest, quarter), playm(0, Rest, eigth), playm(0
           playm(0, Rest, quarter), playm(0, Rest, quarter), playm(0, Rest, eigth), playm(0
            playm(1, Asharp, quarter), susplaymb(1, F, quarter), susplaym(1, F, quarter), su
            susplayme(1, F, quarter), playm(0, Rest, quarter), playm(1, Asharp, eigth), play
           playm(1, D, quarter), susplaymb(1, Asharp, eigth), susplaym(1, Asharp, eigth), su
           playm(0, Rest, quarter), playm(0, Rest, quarter), playm(1, Asharp, eigth), playm(
           playm(1, D, quarter), playm(1, Dsharp, eigth), playm(1, F, eigth), susplaymb(1, A
           playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Res
           playm(1, D, eigth), playm(1, C, eigth), playm(1, Asharp, eigth), susplaymb(1, C,
           playm(1, Asharp, eigth), playm(1, GsharpL, eigth), playm(1, FL, eigth), playm(1,
           playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Res
           playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Rest, eigth), playm(0, Res
             ...
           playm(1, Asharp, quarter), susplaymb(1, F, quarter), susplaym(1, F, quarter), sus
           susplayme(1, F, quarter), playm(0, Rest, quarter), playm(1, Asharp, eigth), playm
           playm(1, D, quarter), susplaymb(1, Asharp, eigth), susplaym(1, Asharp, eigth), su
           playm(0, Rest, quarter), playm(0, Rest, quarter), playm(1, Asharp, eigth), playm(
           playm(1, D, quarter), playm(1, Dsharp, eigth), playm(1, F, eigth), susplaymb(1, A
           play(0, Rest, whole)
         ]
%create chords part
chords = [  play(0, Rest, eigth), play(0, Rest, eigth), play(0, Rest, eigth), play(0, Rest
            play(0, Rest, quarter), play(0, Rest, quarter), play(0, Rest, eigth), play(0, R
            play(0, Rest, quarter), play(0, Rest, quarter), play(0, Rest, eigth), play(0, R
            Bflatq, Bflatq, Aflatq, Aflatq, ...
            Eflatq, Eflatq, play(Rest, 0, eigth), play(Rest, 0, eigth), play(Rest, 0, eigth
            Bflatq, Bflate, Bflate, Aflate, Aflate, Aflate, Aflate,...
            Eflatq, Eflatq, play(Rest, 0, eigth), play(Rest, 0, eigth), play(Rest, 0, eigth
            Bflatq, Bflate, Bflate, Aflate, Aflate, Aflate, Aflate,...
            Eflate, Eflate, Eflate, Eflate, play(Rest, 0, quarter), play(0, Rest, eigth),
```

```
            Bflate, Bflate, Bflate, Bflate, Aflate, Aflatq, Aflate...
            Eflate, Eflate, Eflate, Eflate, Bflath,...
            Bflate, Bflate, Bflate, Bflate, Aflate, Aflatq, Aflate...
            Eflate, Eflate, Eflate Eflate, Bflate, Bflate, Bflatq,...
            ...
            Bflatq, Bflatq, Bflatq, Bflatq,...
            Bflatq, Bflatq, Bflate, Bflate, Bflate, Bflate,...
            Bflatq, Bflate, Bflate, Bflate, Bflate, Bflate, Bflate,...
            Bflatq, Bflatq, Bflate, Bflate, Bflate, Bflate,...
            Bflatq, Bflate, Bflate, Bflate, Bflate, Bflate, Bflate,...
            Bflatw...
            ]
%sum parts into song vector and paly
song = chords + lead + melody;
soundsc(song);

%create and save plots
figure(1);plot(quarter, play(1, C, quarter));

figure(2);plot(quarter, playm(1, C, quarter));

saveas(figure(1),"guitarplot.jpg");
saveas(figure(2), "melodyplot.jpg");

%guitar note
function note = play(volume, frequency, duration)
%initialize ADSR
A = linspace(0, 1, 0.2 * length(duration));
D = linspace(1, 0.6, 0.1 * length(duration));
S = linspace(0.6, 0.6, 0.5 * length(duration));
R = linspace(0.6, 0.2, 0.2 * length(duration) + 2);
ADSR = [A, D, S, R];
%create note with harmonics, ADSR, and frequency/volume modulation
note = ((exp(-length(duration)/100) * volume * cos(2 * pi * frequency * duration)) +
2/3 * (volume * cos(2 * pi * 2 * exp(-length(duration)) * frequency * duration)) +
1.25 * (volume * cos(2 * pi * 3 * exp(-length(duration)) *frequency * duration)) +...
  0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) +  0.125 * (volume * cos(2 * p
0.125 * (volume * cos(2 * pi * 6 * frequency * duration)) + ...
0.02 * (volume * cos(2 * pi * 8 * frequency * duration)) + 0.2 * (volume * cos(2 * pi * 9 *
end


%beginning of held guitar note
function note = susplayb(volume, frequency, duration)
A = linspace(0, 1, 0.2 * length(duration));
D = linspace(1, 0.6, 0.1 * length(duration));
S = linspace(0.6, 0.6, 0.5 * length(duration));
R = linspace(0.6, 0.6, 0.2 * length(duration) + 2);
ADSR = [A, D, S, R];
note = ((exp(-length(duration)/100) * volume * cos(2 * pi * frequency * duration)) +
2/3 * (volume * cos(2 * pi * 2 * exp(-length(duration)) *frequency * duration)) +
1.25 * (volume * cos(2 * pi * 3 * exp(-length(duration)) *frequency * duration)) +...
  0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) +  0.125 * (volume * cos(2 * p
0.125 * (volume * cos(2 * pi * 6 * frequency * duration)) + ...
```

```
0.02 * (volume * cos(2 * pi * 8 * frequency * duration)) + 0.2 * (volume * cos(2 * pi * 9 *
end

%end of held guitar note
function note = susplaye(volume, frequency, duration)
A = linspace(0.6, 0.6, 0.2 * length(duration));
D = linspace(0.6, 0.6, 0.1 * length(duration));
S = linspace(0.6, 0.6, 0.5 * length(duration));
R = linspace(0.6, 0.2, 0.2 * length(duration) + 2);
ADSR = [A, D, S, R];
note = ((exp(-length(duration)/100) * volume * cos(2 * pi * frequency * duration)) +
2/3 * (volume * cos(2 * pi * 2 * exp(-length(duration)) *frequency * duration)) +
1.25 * (volume * cos(2 * pi * 3 * exp(-length(duration)) *frequency * duration)) +...
  0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) +  0.125 * (volume * cos(2 * p
0.125 * (volume * cos(2 * pi * 6 * frequency * duration)) + ...
0.02 * (volume * cos(2 * pi * 8 * frequency * duration)) + 0.2 * (volume * cos(2 * pi * 9 *
end

%middle of held guitar note
function note = susplay(volume, frequency, duration)
volume = 0.6;
note = ((exp(-length(duration)/100) * volume * cos(2 * pi * frequency * duration)) +
2/3 * (volume * cos(2 * pi * 2 * exp(-length(duration)) *frequency * duration)) +
1.25 * (volume * cos(2 * pi * 3 * exp(-length(duration)) *frequency * duration)) +...
  0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) +  0.125 * (volume * cos(2 * p
0.125 * (volume * cos(2 * pi * 6 * frequency * duration)) + ...
0.02 * (volume * cos(2 * pi * 8 * frequency * duration)) + 0.2 * (volume * cos(2 * pi * 9 *
end

%melody note
function note = playm(volume, frequency, duration)
A = linspace(0, 1, 0.2 * length(duration));
D = linspace(1, 0.6, 0.1 * length(duration));
S = linspace(0.6, 0.6, 0.5 * length(duration));
R = linspace(0.6, 0.2, 0.2 * length(duration) + 2);
ADSR = [A, D, S, R];
note = ((volume * cos(2 * pi * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 2 * fr
0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 5
0.03 * (volume * cos(2 * pi * 8 * frequency * duration))) .* ADSR;
end

%beginning of held melody note
function note = susplaymb(volume, frequency, duration)
A = linspace(0, 1, 0.2 * length(duration));
D = linspace(1, 0.6, 0.1 * length(duration));
S = linspace(0.6, 0.6, 0.5 * length(duration));
R = linspace(0.6, 0.6, 0.2 * length(duration) + 2);
ADSR = [A, D, S, R];
note = ((volume * cos(2 * pi * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 2 * fr
0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 5
0.03 * (volume * cos(2 * pi * 8 * frequency * duration))) .* ADSR;
end

%end of held melody note
```

```
function note = susplayme(volume, frequency, duration)
A = linspace(0.6, 0.6, 0.2 * length(duration));
D = linspace(0.6, 0.6, 0.1 * length(duration));
S = linspace(0.6, 0.6, 0.5 * length(duration));
R = linspace(0.6, 0.2, 0.2 * length(duration) + 2);
ADSR = [A, D, S, R];
note = ((volume * cos(2 * pi * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 2 * fr
0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 5
0.03 * (volume * cos(2 * pi * 8 * frequency * duration))) .* ADSR;
end

%middle of held melody note
function note = susplaym(volume, frequency, duration)
note = ((volume * cos(2 * pi * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 2 * fr
0.125 * (volume * cos(2 * pi * 4 * frequency * duration)) + 0.1 * (volume * cos(2 * pi * 5
0.03 * (volume * cos(2 * pi * 8 * frequency * duration)))
end
```

6 Extension

In Figures 1 and 2 I have plotted the guitar and melody cosine function over the period of a quarter note respectively. The ADSR can be seen at work here with the sharp increase followed by a drop to a consistent volume and ending with the decline to near zero. However, there is an obvious difference between the two waves. The guitar wave has a much more confined amplitude and greater variation in its frequency while the melody wave remains relatively constant throughout the time interval. This difference expresses itself obviously in the actual playing of the notes, with the guitar note sounding slightly quieter but with a sharper sound and the melody note being louder with a more consistent and full sound. his ability to view the sound waves graphically could prove to be useful in future iterations of this project. For instance, one could sketch a graph of a sound wave that they think may emulate a certain sound closely, and then derive their equation using that sketch as a reference point. Essentially reverse-engineering the entire process. Although, this would require a very good understanding of how the details of the graph translate to the actual sound.

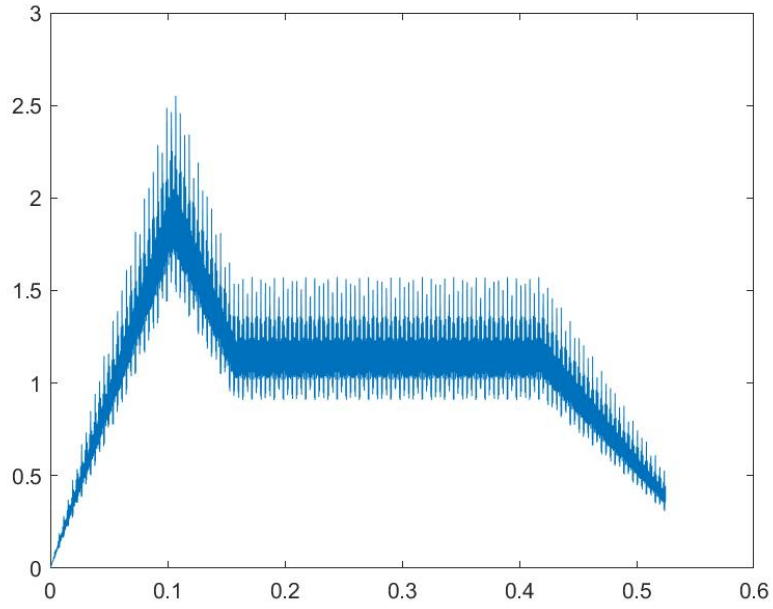

Figure 1: Guitar Note Plot (Amplitude vs Time)

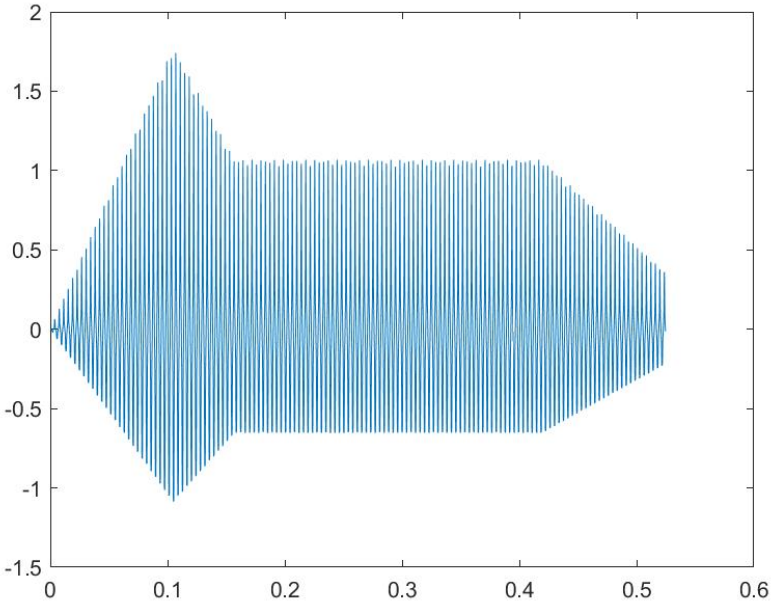

Figure 2: Melody Note Plot(Amplitude vs Time)