Lab 1: Elementary Music Synthesis

University of Washington, Electrical Engineering

EE 341

3 July 2018

|  |  |  |
| --- | --- | --- |
| Kalvin Hallesy 1750416 | Irina Golub 1775424 | Chenglong Li 1429842 |

# 

# 1. Introduction

The purpose of this lab is to compose and digitally improve the sound quality of a segment of music by constructing each note separately in MATLAB and concatenating the notes. To create the song, we first had to create the individual notes with regard to timbre, pitch, and length. This process was described in section 2. Due to the rigidity of the sound created by simply creating uniform notes, we used digital signal processing to make the music sound more realistic. Section 3 shows how the audio quality can be improved by applying dynamic shifts to each note using an ADSR envelope, as if it were played by a physical instrument. To further improve the audio, we overlapped the beginning and ending of each note similar to how a pianist would formulate a song; this is detailed in section 4. This process improves the flow of the song by reducing the choppiness of the notes.

# 2. Building the Song

We used simple tones to compose the segment of music shown in Fig. 1.

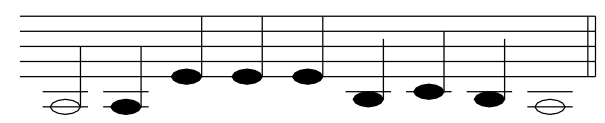
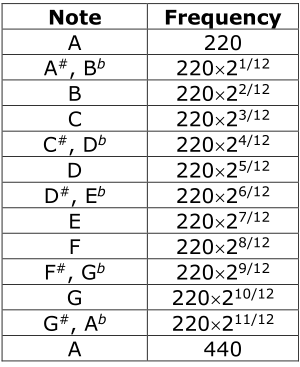


Fig. 1: Musical Score for Scarborough Fair.

The frequency of each musical note depends on the note pitch, as shown in Table 1.

Table 1: Notes in the 220 - 440 Hz octave



Each note is represented by a burst of samples of a sinusoid followed by a shorted period of silence (samples of zeros). The sampling rate is 8 kHz, and each quarter note has a duration of 4,000 samples.

## 2.1 Implementation

First, we defined two vectors with nine entries each where the first entry corresponds with the first note of the song, and so on. The first vector holds the frequency of each note based on Fig. X, and the second vector holds the duration of each note based on its status as a quarter note, half note, or whole note.

We defined the short “rest” in between the each note as a row of 200 zeros. A blank song vector is initiated. Next, a for loop of nine iterations is implemented. Inside the for loop, the next note is generated by calling a function “generateNote” and passing in the note’s frequency, duration, and the sampling rate. After the note has been generated, the note and rest are concatenated to the end of the existing song vector.

The “generateNote” function first creates a time samples vector of the proper duration using the input duration and sampling rate. Using MatLAB’s sin function, the output generated note is a sin of 2pi multiplied by the note frequency and the time samples vector.

The song is the played using MatLAB’s “sound” function with an 8 kHz sampling rate.

## 2.2 Results

The song is successfully played with correct pitch and speed, with no audible distortion. There is a short rest after every note to differentiate between notes.

# 3. Improving Perceived Quality with Volume Variations

The resultant sound of the song created in Section 2 ends up sounding very binary and computer-like. This is because the note is either on or off with no dynamic shift in the note. To combat this, we applied an ADSR envelope to each note which allowed each note to have a similar dynamic profile to the notes of a piano.

The ADSR envelope is divided into four stages: the attack, decay, sustain, and release. During the attack, the amplitude of the tone quickly rises from zero to its maximum value. Immediately after reaching its maximum value, the decay stage begins and the amplitude is rolled down from the peak to an amplitude a little over half of the peak. The sustain stage is the longest and sustains this amplitude value over its duration until entering the release stage where the amplitude tapers down to zero. The ADSR envelope is shown in Fig. 2.

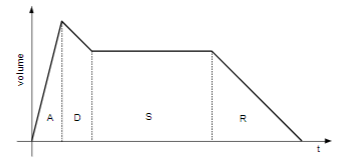


Fig. 2: ADSR envelope

## 3.1 Implementation

To implement the ADSR envelope, we made a separate ADSR function to be called after the creation of each note. This function used four linspace commands, one for each stage of the envelope, to create a seperate line of amplitude scaling values from zero to one with the correct slopes and durations (as percentages of the original note) we deemed appropriate for each stage (these values can be seen in ADSR.m in the appendix). These individual stages were then concatenated and multiplied entrywise with each generated note.

## 3.2 Results

The song was able to play with a much more natural sound; the staccato and harsh cutoff and attack from the original song was replaced by a much more luscious and soft transition between notes. Fig. 3 shows the resultant amplitude vs time plot of the song with ADSR applied. The envelope shape is apparent in each note and is consistent with the sounds that are outputted. Despite these improvements there is still an audible pause and start between each note which is improved upon next.

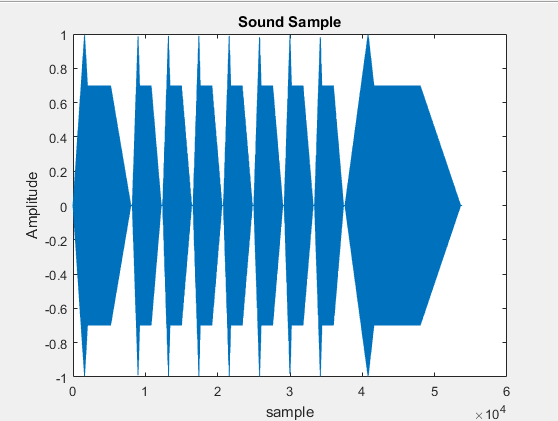


Fig. 3: Sound Sample Plot after Volume Variations.

# 4. Improving Perceived Quality with Overlapping Tones

Another improvement in perceived quality is achieved by overlapping some notes to smooth the flow of the song. Tones can be overlapped by allowing the time regions occupied by subsequent sinusoids to overlap - this results in a smoother song.

## 4.1 Implementation

There are two sections of the song: the existing song, and the next note. In order to overlap the two, we must make both of these the same length and add them together such that the end of the existing song overlaps slightly with the next note. By trying different overlap values, we found that an overlap amount of 12% of the duration of a quarter note. The overlapping strategy is illustrated in Fig. 4.

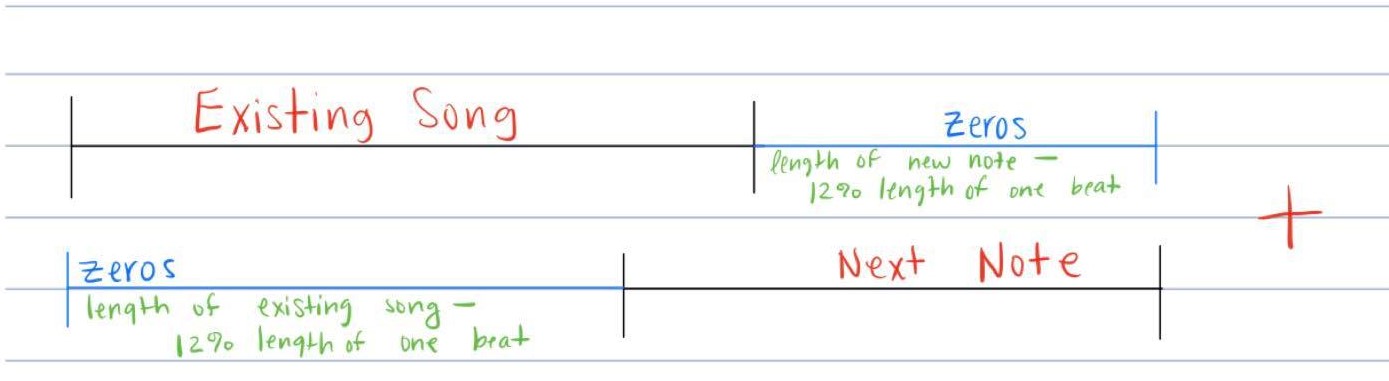


Fig. 4: Implementation of Overlapping Tones.

12% of the existing song vector will be overlapped by the new note. To create a vector of proper length, a zeros vector named ‘overlapVectorEnd’ was defined to be concatenated to the end of the existing song vector. The length of this vector is the length of the new note minus the overlap amount (12% duration of a quarter note). To create a note vector of proper length, a zeros vector named ‘overlapVectorBegin’ was defined to be concatenated to the beginning of the existing note vector. The length of this zero vector is the length of the existing song vector subtracted by the overlap amount. Finally, the appropriate vectors are concatenated and the two portions are added together.

The first note of the song is a special case that will cause the two portions to be different lengths because the existing song vector is initially empty. To account for this, we move the definition of the first note outside the for loop and only run the for loop for the other eight notes.

## 4.3 Results

To ensure that the overlapping did not cause too much distortion, we plotted and observed the sound sample, as shown in Fig. 5.

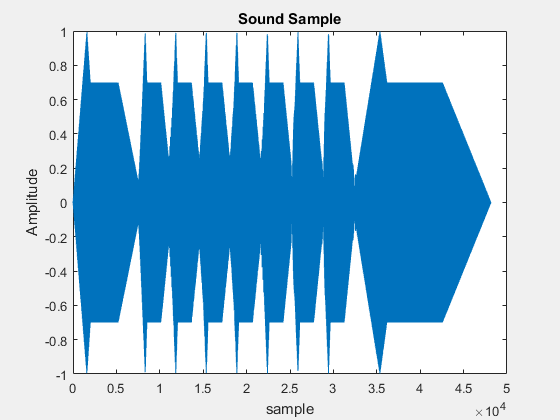


Fig. 5: Sound Sample Plot After Overlapping Tones.

Based on the plot, the overlap can cause some strange frequency spikes to occur in between notes - especially between two dissonant notes. However, this is not noticeable when the sample is played out loud.

Before overlapping was implemented, the song sample was still choppy. Allowing the decaying notes to overlap slightly in time resulted in a song that sounded much smoother. The perceived quality of the sound was, indeed, improved from the previous version.

# 5. Conclusion

A basic tune can be generated digitally by generating individual notes of proper frequency and duration, then concatenating each note. This sounds choppy on its own, so a realistic, dynamic-altering envelope was multiplied to each note to improve the sound quality. Perceived quality was improved further by overlapping some notes, as done by advanced piano players.

## 5.1 Difficulties

The implementation of these three procedures was simple, but we encountered a problem while making the notes overlap. Originally, we designed the code in such a way that 10% of the next note overlapped with the end of the current note. However, we didn’t account for the fact that every note has different lengths - 10% of the last note, a whole note, was a significantly large amount of time compared to the quarter note before it. Thus, the ending of the song sounded very poor. We solved this problem by normalizing the overlap time to specifically hold the value of 10% of the duration of a quarter note. This change yielded the expected results and improved the perceived quality of the piece.

## 5.2 Final Thoughts and Discussion

The final tune was a large improvement of the original in terms of both quality and realisticness which shows the power of even simple signal processing. However, additional improvements can be made to further increase the these traits that were outside the scope of this project. For example, an algorithm that could detect dissonant notes could choose the optimal amount of overlap to minimize dissonance while still overlapping harmonic notes. The ADSR envelopes could also be adjusted to be more continuous by incorporating sigmoids instead of individual lines which would further smooth the sound profile.

# Appendix A

In a zip folder submitted with this report, please find the .m-file for the following sections and functions:

* Part 1 song: “song.m”
* Part 2 song: “Part2Song.m”
* Part 3 song: “Part3Song.m”
* Note-generating function: “generateNote.m”
* ADSR function: “ADSR.m”