

**Project Objective**

The objective of this project is to construct and stimulate physically meaningful signals mathematically using our knowledge in the course ‘Signals and Systems’.

The laboratory will focus on how to generate signals of different frequencies and durations mathematically constructed by using audio playback in MATLAB.

\*Resources used for this laboratory consist of MATLAB software (version 2021b), a sound card installed in a PC to generate and simulate, and a set of headphones to listen to musical creation.

**Table of Contents**

[**1. Background 3**](#_heading=h.akm679i4xvid)

[**2. Note Frequency 4**](#_heading=h.oc7vh13w7bqv)

[**3. Improving perceived quality 5**](#_heading=h.uu0rjo1besmq)

[3.1. Volume variations 5](#_heading=h.h8zza0kdwn9n)

[3.2. Overlapping tones 6](#_heading=h.tx7ibr6904p8)

[**4. Assignment 6**](#_heading=h.9afdko2b40r2)

[4.1. Music Synthesis (25%) 6](#_heading=h.h5cjz6dea64i)

[4.2. Volume Variations (25%) 8](#_heading=h.fjjxwxdkxmjt)

[4.3. Tone Overlapping (25%) 10](#_heading=h.jf3yy161zoep)

[4.4. Favorite Music Synthesis (25%) 11](#_heading=h.g0hda2dif5l3)

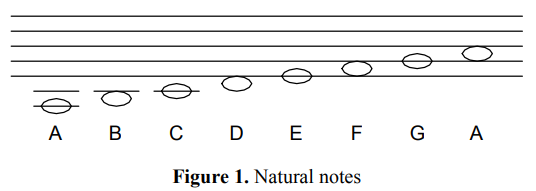
[**5. Conclusions and discussions 12**](#_heading=h.14gyzl6zy0fv)

# **Background**

In this section, we explore how to use simple tones of various frequencies to compose a segment of music. In addition, we will work on improving the perceived quality of the sound we created.

Each musical note can be simply represented by a sinusoid whose frequency depends on the note pitch. In this lab, you will use a sampling rate of8 kHz**.**

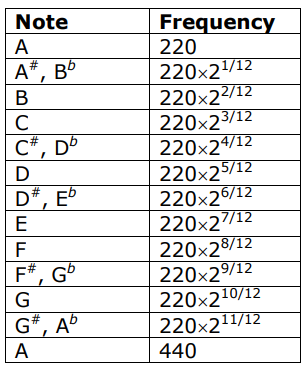
There are seven natural notes: A, B, C, D, E, F and G. After G, we begin again with A. Music is written on a "staff" consisting of five lines with four spaces between the lines. The notes on the staff are written in alphabetical order, the first line is E as shown in Figure 1. Notes can extend above and below the staff. When they do, ledger lines are added.



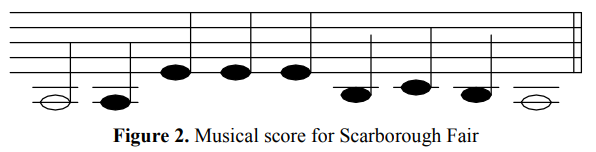
Musical notes are arranged in groups of twelve notes called octaves. The notes that we'll be using for Scarborough Fair are in the octave containing frequencies from 220 Hz to 440 Hz. The twelve notes in each octave are logarithmically spaced in frequency, with each note being of a frequency 21/12 times the frequency of the note of lower frequency.

Thus, a 1-octave pitch shift corresponds to a doubling of the frequencies of the notes in the original octave. Table 1 shows the ordering of notes in the 220-440 octave to be used to synthesize the song, as well as the fundamental frequencies for these notes.

A musical score is essentially a plot of frequencies (notes) on the vertical scale versus time (measures) on the horizontal scale. The musical sequence of notes for the piece we will synthesize is given in Figure 2. The following discussion identifies how musical scores can be mapped to tones of specific pitch and duration.

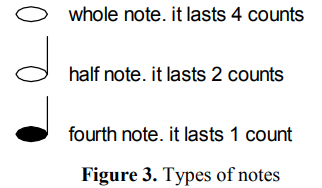


**Table 1**. Notes in the 220 – 440 Hz octave



# **Note Frequency**

In the simplest case, each note may be represented by a burst of a sinusoid followed by a shorter period of silence (a pause). The pauses allow us to distinguish between separate notes of the same pitch. The duration of each note burst is determined by whether the note is a whole note, half note, fourth note, or an eighth note (see **Figure 3**). Obviously, a fourth note has twice the duration of an eighth note, and so on. In this Lab, use the duration of 4,000 samples for 1 count. Therefore, your whole notes should be four times the duration of your fourth notes. The short pause we use to follow each note should be of the same duration regardless of the length of the note. Longer periods of silence that are part of the musical score are indicated by one of more rest symbols.



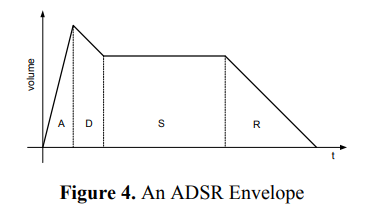
Note that A-G only yields seven notes; the additional notes are due to changes in pitch called sharps (denoted by the symbol #) or flats (denoted by the symbol b) that follows a given note. A sharp increases the pitch by 21/12 and a flat decreases the pitch by 21/12 In the musical score in **Figure 2**, the first half note and fourth note are both A. The next three fourth notes are all E and so on. We can get the fundamental frequencies for these notes from Table 1.

# **Improving perceived quality**

There are many ways of improving the perceived quality of a synthesized sound. Here you will learn about two methods: varying the note volume and overlapping individual tones.

## 3.1. Volume variations

Typically, when a note is played (e.g., using a piano), the volume rises quickly from zero and then decays over time, depending on how hard the key is struck and how long it is depressed. The variation of the volume over time can be divided into four segments: Attack, Decay, Sustain, and Release (ADSR). For a given note, volume changes can be achieved by multiplying a sinusoid by another function called a windowing function. An example of such a function simulating the ADSR effects is shown in *Figure 4*.



## 3.2. Overlapping tones

Another improvement in perceived quality can be achieved by overlapping some notes as done by advanced piano players. As the volume of one note is decaying, another note is played.

Mathematically, this can be accomplished by allowing the time regions occupied by subsequent sinusoids to overlap, hence removing the pause. This will yield a much smoother, less staccato-sounding piece.

# **Assignment**

1. %% Input
2. fs = 8000; %Sampling rate
3. tnote = 4000; %Duration of 4000 samples **for** 1 count
4. whole\_note = 1:4\*tnote; % last 4 count
5. half\_note = 1:2\*tnote; % last 2 count
6. fourth\_note = 1:tnote; % last 1 count

## **4.1. Music Synthesis (25%)**

A sinusoidal function can express the physical meaning of each note (signal). The most common sinusoidal functions are the sine and cosine functions. For this project, sine function is used because it starts at zero, which aligns well with the expectation that a note has no initial sound (or amplitude) before it begins.

A sine function for a continuous signal of an amplitude ‘A’ and of a frequency f, in Hertz is given by

The signal is discretized by sampling at uniform period , by drawing samples at time instants t = n,

X(t)|t=

By dropping , we have the discrete time signal as

, *where is the sampling frequency or rate, and is obtained by taking the reciprocal*

By applying this function for notes into MATLAB, we are able to generate and simulate the note as well as compose the Scarborough Fair notes given in Figure 2.

The code for this part is located in the file ‘MusicSynthesis.m’ that we attach with this report.

* % Notes frequency (220-440Hz)
* pitch = 2^(1/12); % pitch shift
* %Notes used in Figure 2
* A = 220;
* B = A\*(pitch^2);
* C = A\*(pitch^3);
* E = A\*(pitch^7);
* %% Processing
* % Notes in Figure 2
* N1 = sin(2\*pi\*A\*half\_note/fs); % A half\_note
* N2 = sin(2\*pi\*A\*fourth\_note/fs); % A fourth\_note
* N3 = sin(2\*pi\*E\*fourth\_note/fs); % E fourth\_note
* N4 = sin(2\*pi\*B\*fourth\_note/fs); % B fourth\_note
* N5 = sin(2\*pi\*C\*fourth\_note/fs); % C fourth\_note
* N6 = sin(2\*pi\*A\*whole\_note/fs); % A whole\_note

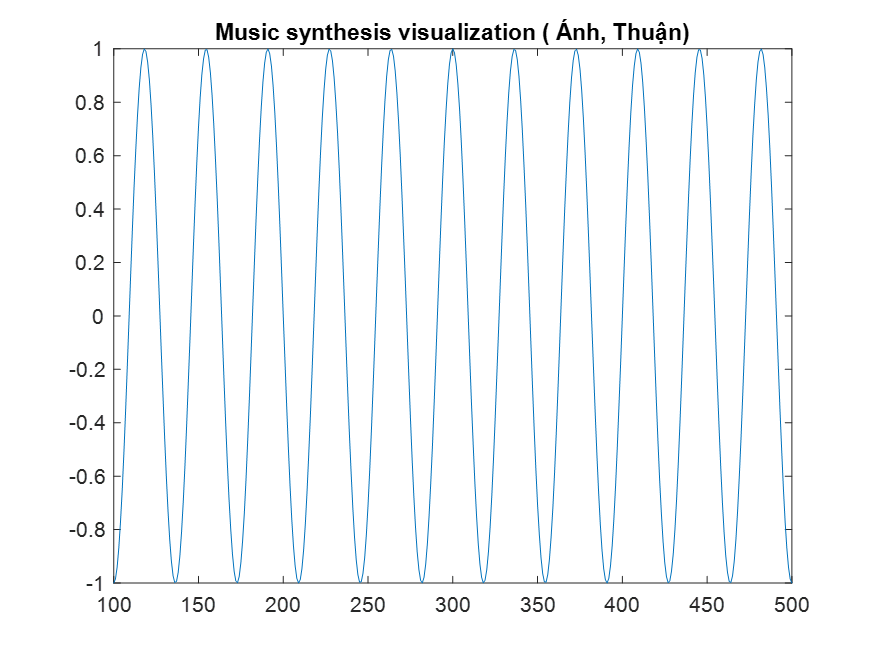


Figure 5: Music synthesis visualization

*Results:*

The synthesized music in MATLAB successfully copied the sequence shown in Figure 2. By applying the concepts of signal generation, we are able to create a simple musical piece, highlighting the relationship between mathematical signals and their physical meaning.

Although the playback of the music sequence represents the fundamental music melody, this part still lacks the smoothness between each note and only plays individual notes (and unchanging volume)with the pause time (see in Figure 5).

## **4.2. Volume Variations (25%)**

As described in *Section 3.1*, varying the note volume can be divided into ADSR segments, which is a way of improving the perceived quality of a synthesized sound.

In order to apply this method, we define an ‘adsr’ function in MATLAB. With the ‘adsr’ function, the duration of Attack, Decay, Sustain, and Release periods are 10%, 10%, 50%, and 30% of the duration of the note length respectively (see in Figure 6). Also, assuming that the volume reaches 1 during attack, comes down to 0.85 during decay and sustains at 0.85 and finally drops to 0. The ‘adsr’ function is given below.

* **function newNode = adsr(note)**
* **L = length(note);**
* **A = linspace(0, 1, 0.1\*L);**
* **D = linspace(1, 0.85, 0.1\*L);**
* **S = linspace(0.85, 0.85, 0.5\*L);**
* **R = linspace(0.85, 0, 0.3\*L);**
* **newNode = note.\*[A, D, S, R];**
* **end**

We also concatenated different functions to model ADSR to stimulate the sound quality with different parameters. The entire process was presented in a ‘VolumeVariation.m’ file, as required. This file contains all the code necessary for synthesizing the music piece with ADSR.

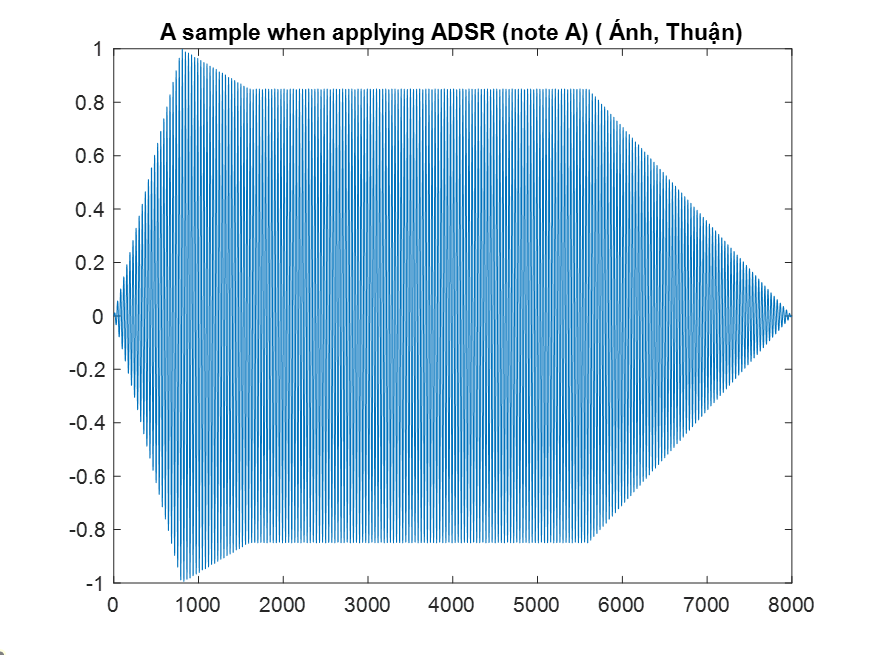


Figure 6: A sample when applying ADSR (note A)

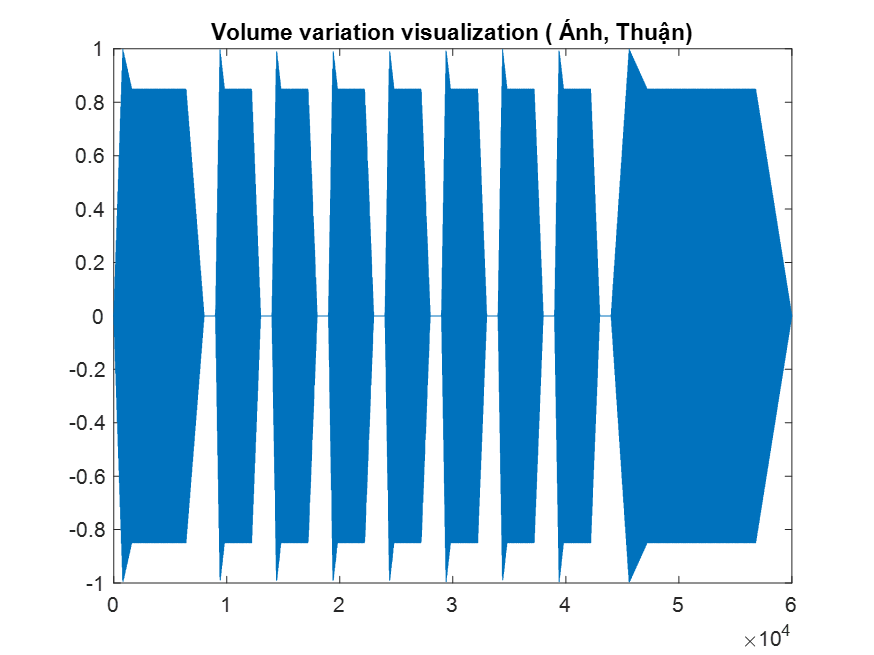


Figure 7: Volume variation visualization

*Results:*

The implementation of volume variations via the ADSR envelope added a more realistic sound to the music piece. The dynamic changes in volume (from Attack, Decay, Sustain, Release) made the piece sound more natural compared to the initial synthesis in *Section 4.1*. The transitions between notes became smoother, and the overall expressiveness of the piece was enhanced.

However, finding the optimal *ADSR parameters* required a considerable amount of experiments, as it *varies*. Additionally, it also still has a delay between each note.

## **4.3. Tone Overlapping (25%)**

As explained in *Section 3.2*, overlapping some notes is a good way to improve the perceived quality of sound, especially in removing the pause between each note.

To implement overlapping tonnes in MATLAB, we allowed the time regions of subsequence sinusoids (note) to overlap. This was achieved by adjusting the end time of one note to extend into the beginning of the next node, effectively eliminating the pause that would typically separate two distinct notes. The implementation function is shown below:

* **function combined = combineWithOverlap(note1, note2, overlap)**
* **overlapPartNote1 = note1(end-round(overlap)+1:end).\* linspace(1, 0, round(overlap));**
* **overlapPartNote2 = note2(1:round(overlap)) .\* linspace(0, 1, round(overlap));**
* **combined = [note1(1:end-round(overlap)) overlapPartNote1+overlapPartNote2 note2(round(overlap)+1:end)];**
* **end**

We experimented with various overlap durations and settled overlap of 10% of duration (from 5% to 15% duration of each note that provided a smooth transition, less staccato-sounding piece, without blurring the distinction between notes, larger than 15% seem not clear and hard to distinguish among each note when they were composed). To simulate code of this part, please look at the code in file ‘ToneOverlapping.m’ that we attach with this report.

*\*\*The music piece in Figure 2 has 3 continuous notes E, which makes it very hard to overlap these notes in the same key in real life. However, in this laboratory, we applied the same overlap duration for notes because it is not required.*

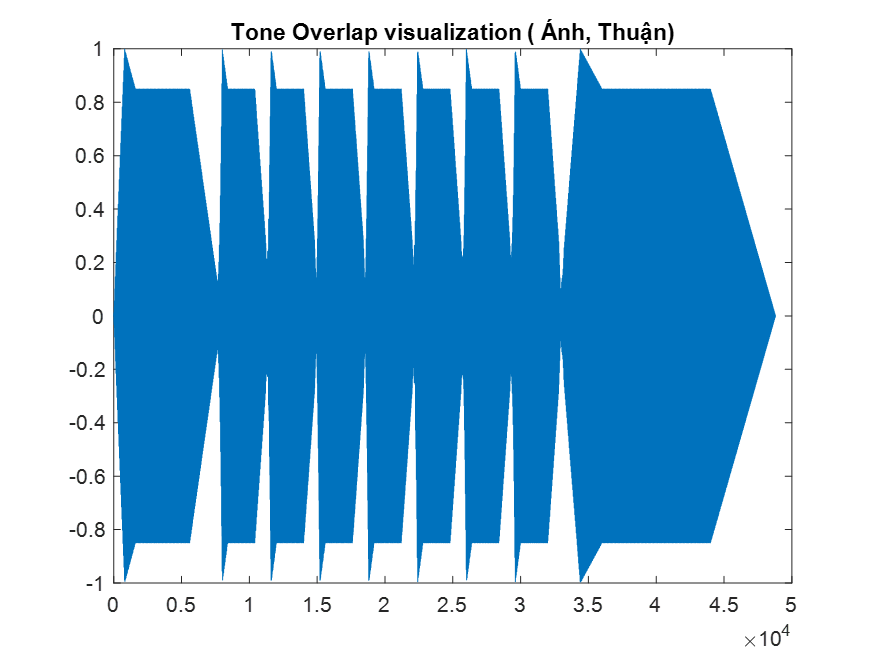
**

Figure 8: Tone Overlap visualization

*Results:*

Overlapping makes the transitions between notes become less abrupt, and look like the music piece is a ‘*live performance*’ by humans. It reduces the mechanical feel, bringing it closer to a natural musical expression.

For this part, we are able to divide signals and combine them together by using knowledge about signals and commands in MATLAB.

## **4.4. Favorite Music Synthesis (25%)**

We repeat those assignments above with the ‘Happy Birthday’ song.

The sheet notes for this song are shown in **Figure 4** and the code implementation for these sheet notes can be seen in ‘HappyBirthday.m’ and the record audio in ‘HappyBirthday.wav’ which is attached together with the report. *Hope it's fun!*



**Figure 9**: Sheet notes for Happy Birthday

# **Conclusions and discussions**

Throughout the Elementary music synthesis laboratory, we are applying our technical skills by illustrating the physical meaning of signals (notes). The lab also provides some knowledge about the basic music synthesis.