Lab 2: Elementary Sound Synthesis

1. Purpose

The purpose of this lab is to: i) implement simple sound synthesis methods, and ii) develop intuitions for the audio impact of different signal transformations.

2. Background

Sound effects, music, speech and many audio signals can be synthesized by computer in a variety of ways. One approach is to generate a simple sound (e.g. a tone or random noise) and then apply various types of transformations to it, such as time scaling, amplitude scaling or filtering. Another approach is to start with natural sounds and modify these. In either case, more complex sounds can be generated by summing up multiple simple sounds, and these can be concatenated to generate sequences of sounds. The lab will use tones and other sounds with amplitude and time modification.

3. Basic Sound Synthesis

Signals can be periodic or aperiodic. In EE235, you learned that periodic signals can be expressed with a Fourier series representation, and real-valued periodic signals can be expressed as a sum of sinusoids. Since the human ear can't hear frequencies much above 20kHz, we can synthesize periodic sounds with a finite number of sinusoids. A musical note is one example of a periodic signal, which we will discuss further in Section 3.1. Aperiodic sounds contain a continuum of frequencies. To synthesize aperiodic sounds in the range of human hearing, we often start with a noise signal and shape it in either the time or frequency domain, as described in Section 3.2. Another approach to synthesizing sounds is modify, mix and concatenate natural sounds (Section 3.3).

3.1 Synthesizing musical notes[1]

Each musical note can be simply represented by a sinusoid whose frequency depends on the note pitch. 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.

[1] *The material related to music synthesis is based on a lab by Professor Virginia Stonick of Oregon State University.*

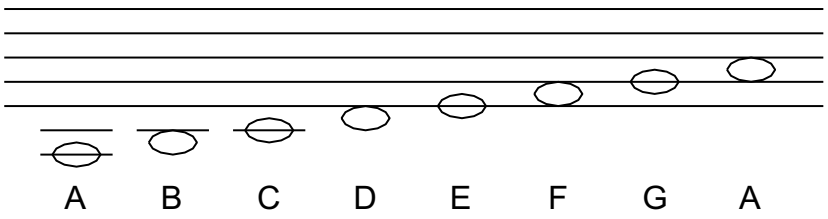

Figure 1. Natural notes.

Musical notes are arranged in groups of twelve notes called *octaves*. The twelve notes in each octave are logarithmically spaced in frequency, with each note being of a frequency $2^{1/12}$ 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 octave from 220Hz to 440Hz, as well as the fundamental frequencies for these notes.

Table 1. Notes in the 220 – 440 Hz octave

Note	Frequency
A	220
$A^{\#}$, B^{b}	$220\times2^{1/12}$
B	$220\times2^{2/12}$
C	$220\times2^{3/12}$
$C^{\#}$, D^{b}	$220\times2^{4/12}$
D	$220\times2^{5/12}$
$D^{\#}$, E^{b}	$220\times2^{6/12}$
E	$220\times2^{7/12}$
F	$220\times2^{8/12}$
$F^{\#}$, G^{b}	$220\times2^{9/12}$
G	$220\times2^{10/12}$
$G^{\#}$, A^{b}	$220\times2^{11/12}$
A	440

The duration of each note burst is determined by whether the note is a whole note, half note, quarter note, etc. (see Figure 2). Obviously, a half note has half the duration of a full note. For this lab, use 0.5 seconds for a whole note. A musical score is essentially a plot of notes on the vertical scale (specifying frequencies), with different forms to indicate note duration, and using the horizontal scale to indicate time ordering. When you have multiple notes lining up vertically, that represents a chord, which is synthesized simply by adding the signals associated with the individual notes.

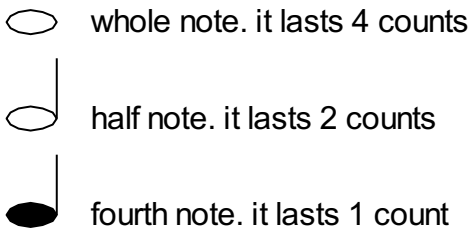

Figure 2. Types of notes.

In the simplest case, each note may be represented by a burst of samples of a sinusoid followed by a shorter period of silence (a sequence of zeros, which are a pause). The pauses allow us to distinguish between separate notes of the same pitch. The short pause you 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. (For more information on music symbols, see https://en.wikipedia.org/wiki/List_of_musical_symbols

3.2 Synthesizing aperiodic sounds

Sounds can be aperiodic simply by being finite in duration, but all computer-based signals are finite-duration. What matters for human perception is whether there is a repeating pattern within a sufficiently long time window – even a few seconds is long enough for something to sound periodic. Very short duration sounds, such as clicks and pops, can be synthesized with pulses of various shapes. A longer duration aperiodic sound can be synthesized by first generating a noise signal, optionally filtering it to adjust the frequency content, and then modifying it with amplitude operations. You can generate the basic noise signal using the randn command in MATLAB. We will use amplitude operations in this lab; you will learn more about filtering later in the course.

3.3 Modifying natural sounds

More natural sounds are often created by starting with actual recorded sounds. For example, a clarinet synthesizer might be based on a few recorded clarinet notes, which are modified through time scaling, amplitude operations, and filtering to create additional notes or even different instruments. Similarly, you could record the pop of a balloon and modify it to produce other sound effects. In Matlab, you can concatenate signals to combine them sequentially (with zero padding to get delays) or add things that you want to mix together.

4. Amplitude Operations

Amplitude scaling is a general concept that is used to improve the perceived naturalness of a synthesized note, but is useful for sound effects as well. The idea is simple: multiply one signal *x[n]* by another *m[n]* to get the modified signal

y[n]=x[n]m[n]. In Matlab, where signals are vectors, you can do an element by element multiply using **y=x.*m**.

One example where this is useful is to approximate the gradual increase and decay in energy of a note produced by a musical instrument. Typically, when a note is played, 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 is 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 function is shown in Figure 3.

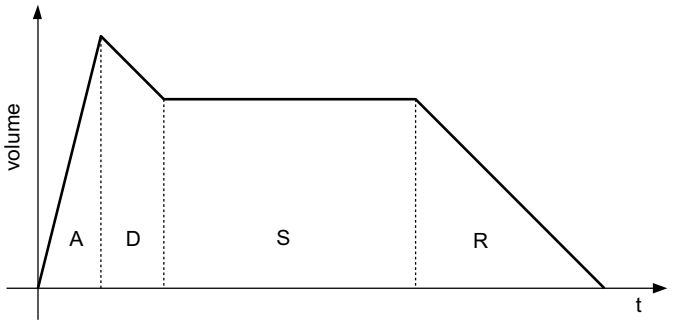

Figure 3. An ADSR Envelope.

Amplitude scaling can involve any function *m[n].* An important example is amplitude modulation, where $m[n]=\cos(2\pi f_0 n)$. In addition to its use in AM radio, multiplying by a cosine is useful to create sound effects, as you will see here.

Another important amplitude operation is addition of signals. You can add multiple harmonics to create a more natural sounding note, add multiple notes at the same time to create a chord, or you can add sounds with a small overlap to create other effects. For example, an improvement in perceived quality can be achieved by overlapping some notes. 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. When combined with ADSR, this will yield a much smoother, less staccato-sounding piece. You can combine many types of signals with addition, including gradually decaying background music at the start of a news announcement or the sum of several voices to create the sound of a party.

5. Time Scaling

A challenge with time scaling for discrete-time signals (compared to continuous-time signals) is that time is defined over the integers. For discrete-time signals, time scaling involves throwing out or adding time samples, for speeding up or slowing down a signal, respectively. (Recall that for periodic sounds, speeding up gives you a higher pitch and slowing down gives a lower pitch, in addition to

being different in length.) Time scaling of the form *y[n]=x[kn]* where *k* is an integer (speeding up) is simple: throw out samples. If *y[n]=x[n/k]* where *k* is an integer (slowing down), you can insert k-1 samples between each original sample and interpolate to get values for the added samples (e.g. using the Matlab **interp1** function).

For rational scaling factors, you can use a combination of up and down sampling. In other words, if y[n]=x[3n/2], you would first slow down by a factor of 2 (referred to as interpolation), and then speed up be a factor of 3 (referred to as decimation). Alternatively, Matlab has a command that will do this for you: **resample**. For example, **resample(x,2,3)** would give you the time-scaled signal for a=3/2.

6. Group Assignment (for in-lab demo)

1. Synthesize an 8-note scale using simple tone bursts using four full notes and four quarter notes (scale in Figure 1 with different note lengths). Use an 8k sampling rate.

2. Improve the quality of the sound with a volume window function (amplitude multiplication). Try concatenating different functions to model ADSR and experiment with allowing a slight overlap in time. Demonstrate for your TA the difference between this version and the unmodified tones in part 1.

3. It is said that when you put a seashell to your ear, you can hear the sound of the ocean. You can create this effect by generating random noise using the randn command in Matlab and multiplying the signal by a low frequency cosine (or shifted cosine, e.g. $1+\cos(2\pi fn)$). Multiply the resulting signal with an exponential decay function to get a sound that fades out. Create such a signal and play it for your TA. You may need to try different cosine frequencies to get an effect that is clearly audible.

4. Download “cat.wav” and “tiger.wav” from the course web page. Use time-scale modification on the tiger to see if you get the effect of a cat or vice versa. Play the resulting signal for the TA.

7. Individual Assignment

1. Synthesize a short segment (a few seconds) of a song you like using tone bursts with your ADSR function and an 8k sampling rate. Look for “musical scores for beginners” to get simple single note scores. (Each person should have a different song.) Upload the file with your report.
2. Download or record an audio file with either music or someone talking and modify it using the same function that you used for the cat/tiger signal. (Each person should have a different signal.) Upload the original and the modified file with your report.

Summary of Assignments:

- In-lab group demonstrations (due during the third week of class)
  - Audio files generated in parts 1-4.
- Individual files to be turned in via Canvas (due the day before you lab in week 3)
  - wav files generated for the individual assignment
  - M-files that were used to create these wav files
  - Brief report (in pdf format) that describes what you implemented in each step, specifics:
    - For (G1): an equation for the discrete-time cosine that corresponds to continuous frequency f_c given sampling frequency f_s.
    - For (G2): a plot of the ASDR window function that you designed, an explanation of how it varies with the length of the note (full vs. half vs. quarter, etc.), and the amount of note overlap you used.
    - For (G3): the cosine frequency and decay factor that you used and your observations about the perceptual effect of varying those parameters.
    - For (G4): the time scaling factor(s) you used, and discussion of the perceptual effect of time scaling the signal that you uploaded.
    - For (I1): an image of the score of your song snippet
    - For (I2): a short description of the original sound, the source of the sound, and what time scaling factor you used.