Laboratory Project M2:
Manipulations of Sound Waveforms

***Abstract*-You will focus on the use of array indexing and functions to modify a sound waveform. You will down-sample the signal, making it sound like chipmunks are singing. You will chop out sections of the music, creating an odd effect. You will chop out words and put them back together in a different order, illustrating speech synthesis. You will apply elementary functions to the music to distort it. You will add shaped noise to the music so it slowly becomes submerged in static. And you will make the music fade out like it does at the end of songs on an mp3 player.**

I. Preparation

Read *Matlab Primer* pages 1-6 to 1-12, 1-17, 1-24 to 1-25, 1-27 to 1-28, and 2-2 to 2-25. Be sure you **bring your earbuds** to all labs so you can listen to sounds produced by your Matlab® programs.

II. Learning Objectives

1) Use indexing arrays to extract or modify segments of a sound sample.
2) Use predefined functions to distort or add noise to the sound sample.

III. Procedure

A. Lab Report

For each part of this lab, you will write a Matlab® script file. At the conclusion of the lab, send your TA an email whose body is the contents of all these script files. That is, put all the script file contents together, put them in the body of the email unless the TA instructs you to do otherwise. Use comments in the script files to identify them and how they work. Before leaving the lab, show the plot in part *C* to the TA and play the sounds created by each of your script files for parts *D* to *I* for the TA.

B. Read Music File into Matlab®

Create a script file called `load_sound.m` for this part of the lab. This script file will read a sound sample into Matlab®. Matlab® has the following built-in sounds that may be easily loaded into the Matlab® workspace: chirp, gong, handel, laughter, splat, and train. For example, `>> load chirp` followed by `>>my_music = y;` reads the chirp sound from a file into an array called `y` and then sets `my_music` equal to `y`.

Listen to the music sample using the `sound()` function in Matlab®. This script file will form the first part of each script file to follow.

C. Plot First 100 samples of Music

Create a script file called `start_sound.m` for this part of the lab. Using the colon operator, create an array (you choose the name for this array) that contains the integers from 1 to 100. Use this array as the index of `my_music` to extract the first 100 values. Plot these first 100 values versus time, label the *x*-axis and *y*-axis appropriately, and add a title to the plot.

Note: Where necessary below, you may use the colon operator, as you have here, to shorten the sound sample. An array of 8000 samples corresponds to one second of sound. If you use a sample that is too short, you may not be able to hear it.

Note: You may wish to refer to Lab M0 on the course website for instructions on plotting.

D. Down Sampling

Create a script file called `down_samp.m` for this part of the lab. Using an integer array for indexing, create a new array that is the music array with every other sample deleted. Hints:

1) To delete a sample, set its value to [ ].
2) use the colon operator to create indices of values you wish to delete. Use the colon indexing on the left side of the equals sign.

Play the sound using the `sound()` function and, in a comment, describe how it sounds.

E. Chopping

Create a script file called `chop_it.m` for this part of the lab. Using an integer array for indexing, set segments of (a copy of) `my_music` array to zero. The `my_music` array will have n samples on following by n samples zeroed out. Use the last two digits of your student ID number as the value of n. This pattern then repeats to the end of the array.

For example, if last two digits of your student IC number are 04 the original and chopped array would appear as follows:

```
my_music      = [5, 2, -3, -7, -4, 6, 9, 12, 8, 3, -2, -5, -1, 5, 8, 3, 1, -1, ...]
my_music_chop = [5, 2, -3, -7,  0, 0, 0,  0, 8, 3, -2, -5,  0, 0, 0, 0, 1, -1, ...]
```

The key idea is to create an array of ones and zeros to multiply the signal element-by-element. Where the original signal is multiplied by one, its value is preserved. Where the original signal is multiplied by zero, the result is zero.

How to create the array of ones and zeros:

1) Create an array of the first n ones and n zeros. Use square brackets, [ ], and a `ones` command and a `zeros` command to do this.
2) Use the `repmat` function to create the repeating array of ones and zeros that you will multiply `my_music` with element-by-element. `repmat` is like a rubber stamp that makes copies of an array to create a larger array. You tell `repmat` what array to duplicate and how many copies of that array you want vertically and horizontally.

   Example: (replicating a 2 x 2 matrix M = [1, 2; 3, 5])

```
>> big_array = repmat(M,2,3) % 2 copies of M vertically, 3 copies horizontally
big_array =

     1     2     1     2     1     2
     3     5     3     5     3     5
     1     2     1     2     1     2
     3     5     3     5     3     5
```

   Type "`help repmat`" at the Matlab® prompt to learn more about `repmat`.

Play the chopped sound using the `sound()` function and, in a comment, describe how it sounds.

F. Concatenated speech

Create a script file called `scramdel.m` for this part of the lab. Use the `handel` sound. Using the method described in part *B*, above, extract portions of the music corresponding to several syllables of words being sung and store the syllables in separate arrays. Concatenate them in a different order to create a nonsense word. Listen to your sound and describe, in comments, what reordering was done and how it sounds.

G. Nonlinear function

Create a script file called `distort_it.m` for this part of the lab. Apply a weighted sum, (e.g., `0.2 * func1(my_music) + 0.3 * func2(my_music)`), of two (but not just the first two) of the following functions (or any other functions you may discover in Matlab®) to every value in the `my_music` array to create a new sound (stored in an array) and listen to the result.

```
.^3 % cube every sample value  power()  abs()  exp()  sign()  sinh()  erf().^2
```

For a list of interesting functions you might use, type "`help elfun`" at the >> Matlab® command prompt. Type "`help power`", etc. at the Matlab® prompt to learn more about these functions. In a comment, describe how your distorted waveform sounds.

H. Noise

Create a script file called `noise_shaped.m` for this part of the lab. Your task is to create an array of noise samples that is as long as your `my_music` array, multiply the noise array by another array containing samples from one half-cycle of a sinusoid (that stretches over the length of the sound), causing the noise to grow in loudness and fade out again. (The trick here is to figure out the frequency to use in `sin(2*pi*frequency*t)`). You will add this noise to the `my_music` array. That is, you will create an array equal to `my_music` plus the shaped random noise.

If your student ID # is even, use the `rand()` function to generate your initial array of noise samples. If your student # is odd, use the `randn()` function to generate your initial array of noise samples. To control the loudness of the noise, multiply the noise array by a constant of your choice (found by listening and experimenting) that is less than one times the maximum value of the `my_music` array. Use the `max()` function to determine this maximum value.

I. Fade

Create a script file called `fade_out.m` for this part of the lab. Your task is to create an array called `fade` containing samples from a decaying exponential that, when multiplied by your music array, will create a sound that fades out.

The decaying exponential array should start at a value of 1.0 and decay to a value of 0.01 in five seconds. Five seconds corresponds to 40,000 samples at a sampling rate of 8000 samples per second. If we assume that our exponentially-decaying waveform is calculated by applying an `exp`onential function an array of integers, [0, 1, 2, ...], then we want to first multiply the integers by a scaling factor, *k*, so that exp(k*40000) = 0.01 for the proper fade. In other words, we must solve the following equation for *k*. Note that *k* will be negative.

$$e^{k(40000)} = 0.01$$

Once you have solved for *k*, create the `fade` array as follows:

1) create an array of values from 1 to the number of samples in your `my_music` array,
2) multiply the array by *k*, and
3) apply the `exp()` function to the result to get the `fade` array.

Multiply `my_music` by this `fade` array (point-by-point) to create a music sample that fades out.

Play the sound using the `sound()` function and comment in your script file on how it sounds. In particular, comment on whether the sound seems to fade out at a steady rate. Does an exponential fade sound like a linear fade?

REF: [1] The Mathworks, Inc, *Matlab® Primer,* Natick, MA: The Mathworks, Inc, 2012.