Laboratory Project M4: Creating Sound Effects

***Abstract*-You will create script and function files to implement various sound effects based on modifying samples or a sound's spectrum. You will also create a unique sound effect of your own design.**

I. Preparation

Read Matlab Primer Chapter 5 pages 5-1 to 5-13, and 5-16 to 5-17.

II. Learning Objectives

1) Use a switch command
2) Use for loops
3) Write function files
4) Use `fft` (Fast Fourier Transform) command to create spectrum of sound
5) Understand the effects of modifying a spectrum

III. Procedure

A. Lab Report

In this lab, you will write several script and function files. At the conclusion of the lab, send your TA an email whose body is the contents of all these script files and other information as specified below. That is, put all the script and function file contents together, and put them in the body of the email, unless the TA instructs you to do otherwise. Use comments in your files to identify them and how they work.

Show your files (lab report) to your TA before leaving the lab.

B. Sound Effects based on Magnitudes of Samples

Create a script file called `clipping.m` for this part of the lab. This script file will process a sound waveform based on the values of the samples in the waveform and apply one of several different sound effects, depending on the value of a string variable called `effect`. Write `clipping.m` to do the following:

1) Load the `handel` sound, (which by default will be placed in the variable `y`).
2) Shorten `y` to two seconds of sound (# samples = 2 sec * 8000 samples/sec).
3) Use a switch statement that branches according to the string variable called `effect` as follows.
4) If the value of `effect` is `'clip'`, set all sound samples greater than 0.1 to 0.1 and samples less than –0.1 to –0.1, and play the sound. (This operation clips off peaks.)
5) If the value of `effect` is `'squish'`, delete all sound samples that are greater than 0.2 or less than –0.2, and play the sound. Use a single instruction for this step. If you try to use a `for` loop, the indexing into the `y` array will change every time a sample is deleted, causing the bookkeeping to become a nightmare.
6) If the value of `effect` is `'rectify'`, set all sound samples less than 0 to 0, and play the sound.
7) If the value of `effect` is `'sine_mask'`, set samples to the larger (in magnitude) of two waveforms: the sound in `y`, and a 100 Hz sinusoid of magnitude 0.2.

8) If the value of `effect` is not recognized as one of the above terms, display the message "unsupported effect".

Test each effect and use the `listen` command to play the sounds.

C. Sound Effect based on Median Filter

Create a script file called `medfilter.m` that uses a `for` loop to step through a sound snippet already stored in `y` and turn it into a stair-step waveform. The `medfilter` script operates as follows:

1) At the command prompt (rather than in the script file), create a variable called `nsamps` that is equal to the last digit of your student ID number, unless the last digit is less than two. If the last digit is less than two, use ten plus the last digit of your student ID number. The value of `nsamps` specifies the block size for processing samples of the sound. The value of `nsamps` also specifies the length of the stair-steps in final waveform. Your script file will assume the value of `nsamps` is already defined when the script file is run.
2) In `medfilter.m`, start by checking for `nsamp < 2`, which is an error. If this error occurs, `medfilter.m` should display an appropriate message and return.
3) After error checking in `medfilter.m`, use a for loop to extract the next `nsamps` of the sound waveform in `y` each time through the loop. An easy way to do this is to use the colon operator to create the `for` loop indexing variable as an array from 1 to the length of `y` and incremented by `nsamps`. The indexing variable will then contain the starting address in y of the current block being processed.
4) Replace the current block in `y` with the mean value (i.e., average value) of the current block in y. That is, replace every value in the current block with the average value of the block. This process will cause the final waveform to have stair-steps that are `nsamps` long.

Note: it may be helpful to use concatenation such as `yout = [yout, ymed];` to add samples at the end of an output array each time through the `for` loop. In this case, `ymed` would be `nsamps` of computed in step (4), above. At the end of the script file, you can set `y` equal to `yout`. Note that this approach requires a statement such as `yout = []` before starting the `for` loop.

D. Sound Effect based on Repeated Samples

Create a function file (not a script file) called `rep.m` that has the following input variables:

```
% invec          input sound array
% start_samp     sample number where repeated sound will start
% rep_size       size in samples of repeated sample
```

`rep.m` will output a variable called `outvec` containing the sound array in `invec` with a section of repeated samples inserted in it. That is, the function should extract a section of `invec` of lenght `rep_size` and then insert a copy of that extracted section at the end of the section. This will make the sound longer, and the result will be a reverberation or stuttering effect. Experiment with `rep.m` to find an interesting value of `rep_size`. Play the sound for your TA.

E. Sound Effect based on Echoes

Create a function file (not a script file) called `echoes.m` that has the following input variables:

```
% invec          input sound array
% echo_delay     number of samples after which echo starts
% echo_size      loudness of echo relative to original (1.0 means equally loud)
```

`echoes.m` will output a variable called `outvec` containing the sound array with echoes in it. The output sound is constructed by adding copies of the original sound scaled by `echo_size` and delayed by `echo_delay` samples. Recall that 8000 samples equals one second. To avoid creating an overly loud sound, scale the final output sound so the median value is the same as it was before the echoes were added. That is, multiply the entire waveform by a constant so that the average or mean value of the waveform is the same as it was before echoes were added.

Each successive echo will be created by multiplying the previous echoe by `echo_size`. Thus, each echo is smaller than the last, and the delay for the echo increases by `echo_delay`. Note that the length of the final sound should be the same as the original sound. Eventually, the echoes will be dealyed so much that they no longer overlap the original sound and may be ignored.

F. Sound Effect based on Spectrum

Create a function file (not a script file) called `spechange.m` that has the following input variable:

```
% invec          input sound array
```

`spechange.m` will output a variable called `outvec` containing the input sound array with an altered spectrum. Your function must first compute the `fft` (Fast Fourier Transform) or spectrum of the input sound array. Then your function will alter the spectrum in some way. The spectrum presents the sound in terms of its frequency content. You may think of the spectrum as phasors represnting the amount of each frequency in the signal from 0 to one-half the sampling rate of 8000 Hz. The first half of the spectrum computed by the `fft` command contains all the frequency information for the signal. The second half is redundant; it is equal to complex conjugate of the first half in reverse order. Altering samples at the beginning and end of the spectrum affects the low frequencies in the sound. Altering samples in the middle of the spectrum affects the high frequencies in the sound.

The spectrum has complex values that are much like phasors. They encode the magnitudes and phase shifts of the sinusoids in the sound. They are in the rectangular format `a + j*b`, however. You can convert to magnitude and phase by using `sqrt` and `atan2`. This may be helpful if you decide to process the spectrum based on magnitudes or phase shifts.

After you have altered the spectrum, use `ifft` to convert the spectrum back into a time-domain waveform. Since the `fft` is complex and your alterations to the sound's spectrum may not result in `ifft` being real, you will probably need to apply the `real` function to your final sound array. Note: when plotting complex values, you can apply the `abs` function to extract magnitude.

Be creative in the ways you alter the spectrum of the sound. You may alter the spectrum in any way you like as you try to create an interesting result.

G. Original Sound Effect

Create a function file (not a script file) called `effect.m` that implements a unique sound effect of your own design. Your sound effect function may have whatever arguments you wish after the first argument, `invec`, which is the initial sound:

```
% invec          input sound array
```

`effect.m` will output a variable called `outvec` containing the input sound with your sound effect applied to it. Note that your function must work with any input sound. Include comments in your file to describe the purpose of each line of code.

You will likely finish your sound effect outside the lab period. When it is finished, send it to your TA in whatever format the TA specifies.

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