Multimedia
Module No: CM3106
Laboratory Worksheet Lab 3 (Week 4): MATLAB Digital Audio Synthesis (Part 1)

Prof. D. Marshall

After working through this worksheet you should be familiar with:

- The basic theories of digital audio synthesis: Subtractive, Additive and FM Synthesis
- The basic implementation of digital synthesis (Subtractive, Additive and FM Synthesis) techniques in MATLAB.

None of the work here is part of the assessed coursework for this module ALTHOUGH many of the exercises below will help in parts of of your solution for the assessed coursework

MATLAB Basic Digital Audio Synthesis

Zip file for all Digital Audio Synthesis examples is available here.

1. **Subtractive Synthesis**: Create a square wave and sawtooth wave signal at a given frequency. Now apply a low pass/high pass filter/bandpass filter at different cut-off/centre frequencies. to this data. Listen to and also display the waveforms you produce. (Hint: Download the *subtract_synth.m* demo from the lectures and simply modify some parameters. Also look at the *synth.m* function it uses to create assorted waveforms.)

2. **Subtractive Synthesis**: Create a *white noise* signal (randomised data). Now apply a low pass/high pass filter/bandpass filter at different cut-off/centre frequencies. to this data. Listen to and also display the waveforms you produce. Also modulate the cut-off frequencies in the filters. Using these techniques try simulate the sound of aircraft, ocean waves and/or wind.

3. **ADSR**: Implement an ADSR MATLAB function that takes in as input parameters: an input waveform, the attack, delay, sustain and release parameters in milliseconds (or sample duration) and gain parameter as a percentage of the overall gain and *outputs* an waveform with the ADSR envelope applied.
(***Note: This functionality is required as part of the assessed coursework requirements also***).

4. **Additive Synthesis**: Using the MATLAB lecture example code *additive_synth.m* as a basis experiment with adding square, sawtooth and sine waves together to create some new waveforms.

5. **Additive Synthesis**: Perform additive synthesis in ***Fourier space***. Take two simple waveforms (not all sines of cosines) and/or some audio samples and compute their Fourier transforms. Filter the Fourier transformed waveforms into discrete frequency bands, for example, 0-200 Hz,200-300Hz, 400-500Hz etc. or in bands which are steps up from the input waveform frequency.

   Add only certain bands together: write a MATLAB function that for each frequency band it applies some weight (gain) and adds the two

frequency space waveforms together. Weights can be 0 (i.e. filter) or any other values so that different bands can be mixed in any linear combination.

Inverse Fourier transform the results and listen (and display) to the outputs. Try a few variations on the above.

6. **FM Synthesis** Using the lecture example MATLAB code *fm_eg.m* change the following parameters as given in the MATLAB code fragment below:

```
fs = 22050;
T = 1/fs;
dur = 7.0;
t = 0:T:dur;
T60 = 1.0;
env = 0.95*exp(-t/T60);

% FM parameters
fc = 200;
fm = 280;
Imax = 10;
I = Imax.*env;
```

Apply the basic FM synthesis equation as given in *fm_eg.m*. What does the output sound like?

7. **FM Synthesis** Using the lecture example MATLAB code *fm_eg.m* change the following parameters as given in the MATLAB code fragment below:

```
fs = 22050;
T = 1/fs;
dur = 0.2;
t = 0:T:dur;
T60 = 0.1*dur;
env1 = exp(-t/T60);
env2 = 1.0 - t./(0.2*dur);
env2 = env2 .* (1.0 + sign(env2))/2.0;
```

```
% FM parameters
fc = 80;
fm = 55;
Imax = 25;
I = Imax.*env2;
```

Apply the basic FM synthesis equation as given in *fm_eg.m* but **modified** so that `env1` modifies the amplitude of the basic FM equation, i.e.:

```
y = env1.*sin(2*pi*fc*t + I.*sin(2*pi*fm*t));
```

What does the output, `y`, sound like?