EE386 Lab 2 - Fourier, Beat, Chirp, Tones, and Bells

Extra Credit Section

Dec. 16, 2018

Lab directed by Chris Hirunthanakorn

Lab conducted by Paco Ellaga

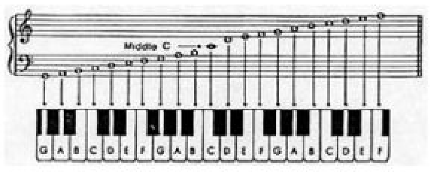
CSULB SID: 009602331

Introduction

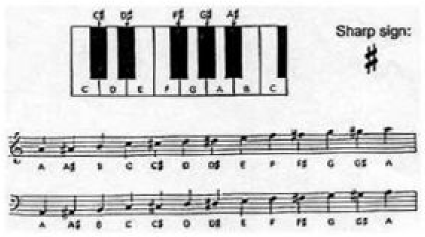
In the extra credit portion of Lab 2, we were tasked with using the ADSR (Attack -Decay-Sustain-Release) Envelope Generation code (adsr_gen.m) to simulate notes played on keyboard to play the C major scale. This set of code shown in ***Section 1*** in the ***Appendix***, is the code given in Part 3 instructions of Lab 2 and had to be manipulated to complete this section. I, personally, did not actually play the C scale in the exact instructions as intended and ended up reworked the timing and structure of the ADSR_gen code into a finalized separate m.file that plays the first 14 measures of the 1982 song "Africa" by Toto which also is in a different scale. Overall, it does exactly fulfill exact instructions, due to scale differences, it fulfill the objective to play a multitude of various notes.

Procedure

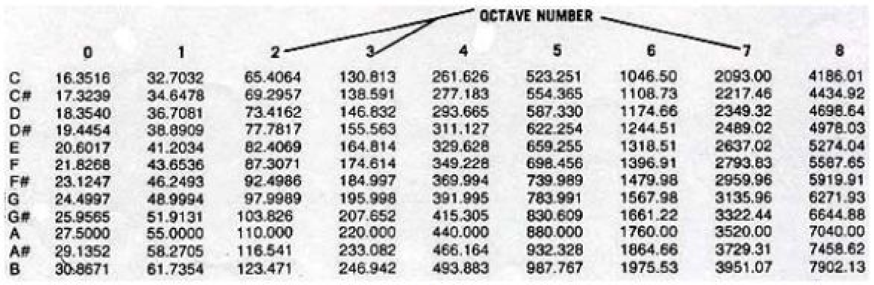
To begin, the given adsr_gen.m file needed to be broken down line by line to prove that the function could actually recreate the physical action of playing a note. After full analysis (fully explained in the ***Analysis*** section), a restudy of configuring notes to actual numerical frequencies had to be taken into account. This was taken through studying the picture provided in ***Figure 1***.



(a)


C♯ D♯ F♯ G♯ A♯
C D E F G A B C
Sharp sign:
♯
A A♯ B C C♯ D D♯ E F F♯ G G♯ A
A A♯ B C C♯ D D♯ E F F♯ G G♯ A

(b)



	OCTAVE NUMBER								
	0	1	2	3	4	5	6	7	8
C	16.3516	32.7032	65.4064	130.813	261.626	523.251	1046.50	2093.00	4186.01
C#	17.3239	34.6478	69.2957	138.591	277.183	554.365	1108.73	2217.46	4434.92
D	18.3540	36.7081	73.4162	146.832	293.665	587.330	1174.66	2349.32	4698.64
D#	19.4454	38.8909	77.7817	155.563	311.127	622.254	1244.51	2489.02	4978.03
E	20.6017	41.2034	82.4069	164.814	329.628	659.255	1318.51	2637.02	5274.04
F	21.8268	43.6536	87.3071	174.614	349.228	698.456	1396.91	2793.83	5587.65
F#	23.1247	46.2493	92.4986	184.997	369.994	739.989	1479.98	2959.96	5919.91
G	24.4997	48.9994	97.9989	195.998	391.995	783.991	1567.98	3135.96	6271.93
G#	25.9565	51.9131	103.826	207.652	415.305	830.609	1661.22	3322.44	6644.88
A	27.5000	55.0000	110.000	220.000	440.000	880.000	1760.00	3520.00	7040.00
A#	29.1352	58.2705	116.541	233.082	466.164	932.328	1864.66	3729.31	7458.62
B	30.8671	61.7354	123.471	246.942	493.883	987.767	1975.53	3951.07	7902.13

(c)

Figure 1: (a) Octave codes, (b) Note codes, (c) Frequency data.

After that, to attain the proper key structure of Toto's "Africa", sheet music was needed to be acquired or developed. The sheet music acquired was originally uploaded by user, Pinkpeppermint10, on musescore.com on April 26, 2016 and ended up becoming the basis for the code that ended up being written in this project. The score was set to be for a piano and flute duo and the notes generated on MATLAB was set to the piano treble clef () part. As for the music scales used for this project, Toto's song "Africa" was originally written to be played in the A-major scale (***Figure 2***), but the sheet music was written in the E-major scale (***Figure 3***), and (although it was known that) the project was to be written in C-major scale (***Figure 4***). Due wanting to play "Africa", it had to be changed due to the scale sounding too flat when in the C major scale. The only differences that was modified are that the notes C, D, F, and G are instead played as their sharp counterparts so it be played in the E-major scale, whereas they are not sharp in the C-major scale.

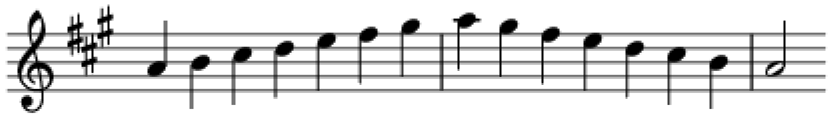


Figure 2: *Key signature for the A-major scale*

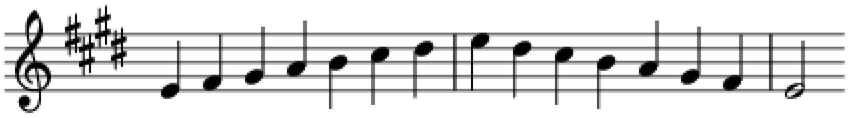


Figure 3: *Key signature for the E-major scale*

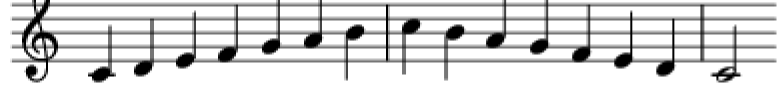
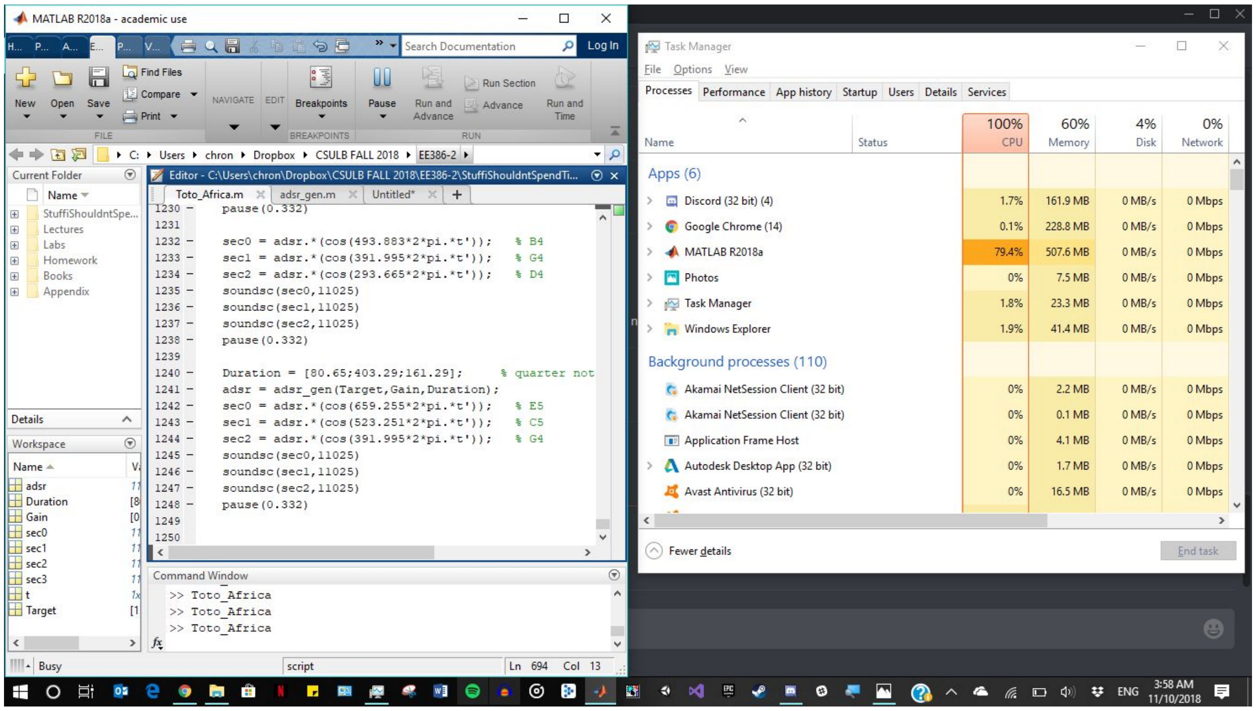


Figure 4: *Key signature for the C-major scale*

After all the code was written, to fully play the file, the adsr_gen.m file needed to be added to the project. This will be all that is needed is to run the Africa.m file and listen, as the response is all auditory. The duration of the audio projection from the code is about 50 seconds and will play to the 14th measure of the song, from the beginning. This single run will only play it one time through. If the user wanted to save a copy of the recording, the *saveas()* command could be used at the very end. Loops could have been a valid way to replay sections, but trial and error, MATLAB would crash my laptop. Through an extra look of the whole processing of the code, the task manager would show that when the code is played without loops, MATLAB will be immediately forcing the CPU of my laptop to 100% for the entire duration of the audio projection. Below is the snapshot ***Figure 5***, when the code was in the process of making without the loops. A figure of the playing a looped could not be provided due the whole computer just instantly freezing requiring a force restart to proceed any further.



```
    pause(0.332)

    sec0 = adsr.*(cos(493.883*2*pi.*t'));    % B4
    sec1 = adsr.*(cos(391.995*2*pi.*t'));    % G4
    sec2 = adsr.*(cos(293.665*2*pi.*t'));    % D4
    soundsc(sec0,11025)
    soundsc(sec1,11025)
    soundsc(sec2,11025)
    pause(0.332)

    Duration = [80.65;403.29;161.29];      % quarter not
    adsr = adsr_gen(Target,Gain,Duration);
    sec0 = adsr.*(cos(659.255*2*pi.*t'));    % E5
    sec1 = adsr.*(cos(523.251*2*pi.*t'));    % C5
    sec2 = adsr.*(cos(391.995*2*pi.*t'));    % G4
    soundsc(sec0,11025)
    soundsc(sec1,11025)
    soundsc(sec2,11025)
    pause(0.332)
```

```
>> Toto_Africa
>> Toto_Africa
>> Toto_Africa
```

Name	Status	100% CPU	60% Memory	4% Disk	0% Network
Apps (6)					
Discord (32 bit) (4)		1.7%	161.9 MB	0 MB/s	0 Mbps
Google Chrome (14)		0.1%	228.8 MB	0 MB/s	0 Mbps
MATLAB R2018a		79.4%	507.6 MB	0 MB/s	0 Mbps
Photos		0%	7.5 MB	0 MB/s	0 Mbps
Task Manager		1.8%	23.3 MB	0 MB/s	0 Mbps
Windows Explorer		1.9%	41.4 MB	0 MB/s	0 Mbps
Background processes (110)					
Akamai NetSession Client (32 bit)		0%	2.2 MB	0 MB/s	0 Mbps
Akamai NetSession Client (32 bit)		0%	0.1 MB	0 MB/s	0 Mbps
Application Frame Host		0%	4.1 MB	0 MB/s	0 Mbps
Autodesk Desktop App (32 bit)		0%	1.7 MB	0 MB/s	0 Mbps
Avast Antivirus (32 bit)		0%	16.5 MB	0 MB/s	0 Mbps

Figure 5*: Snapshot proving that my laptop can barely withstand the intensity of the code that is being generated.*

<u>**Analysis**</u>

NOTE: For reference, ***Section 1* in the ***Appendix***, has the original asdr_gen code. ***Section 2*** contains the code for Toto_Africa.m, but will request for the reader to see the externally attached .m-file in the dropbox, as it is 1248 lines of code and will extend the length of the report by an extra 44 pages if were directly inputted in the ***Appendix section*****

To begin analysis of the adsr_gen.m code, the attack, decay, sustain, and release phases were individually broken down. It was found that the code only needed to calibrate the attack, sustain, and release phases. The decay phases were naturally implemented into the attack and sustain phases using this following line of code:

a(n) = target(1)*gain(1) + (1.0 - gain(1))*a(n-1);

Equation 1: *How to calculate decay into the attack (target(1)) and sustain (target(2)) phases*

Each phase are set with their own target volume and each of their own portion of time is based over the complete time frame restricted by the function's parameters (***Figure 6***). The initial parameters of the code was set to revolve over the course of one whole second. As each phase ended, the execution of the next was immediately activated and applied until the whole function ended and produced the specific frequency that it was encoded with.

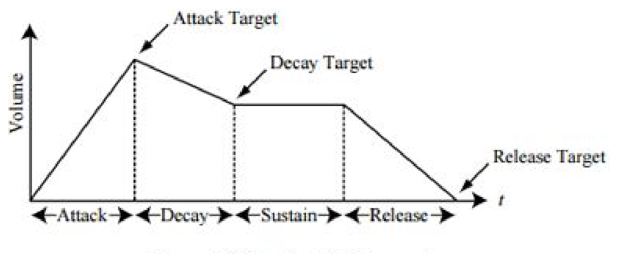


Figure 2: Classic ADSR envelope

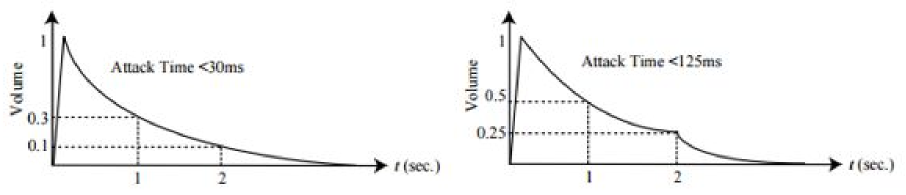


Figure 3: (a) ADSR envelopes for guitar, (b) ADSR envelopes for piano.

Figure 6: *The figure visually encompassing the parameters of the adsr_gen.m file code*

The distinctly audible parts are the attack and the sustain portions, from which they were manipulated to accommodate "Africa"'s tempo in beats per minute (BPM) of 93 BPM, where on a 4/4 scale, one quarter note equates to one beat. Using this information, it was applied directly for the construction of the notes. As to actually developing the notes, this following lines of code from Lab 2 - Part 3 was used:

```
Target = [1-eps;0.25;0]
Gain = [0.005;0.0004;0.00075]
Duration = [125;625;250]
adsr = adsr_gen(target,gain,duration);
```

Equation 2: *Parameters to play a single note*

This provided the means to translate ratios for the duration of the notes from 1 second to have it play in 93 BPM. By starting with the duration to be initially be 120 BPM, it would be modified through ratios to match the notes being played at 93 BPM. The duration ratio of 125;25;250 for a 120 BPM was set to a half note and then applied with a 120:93 ratio. For all of the half notes to be played at 93 BPM, the duration will have to be set to 161.3;806.58;322.58. As for the other note length conversions, they are shown in ***Appendix, Section 2 (lines 18-37)***.

From there, the *sec0, sec1,* & *sec2* variables were set with the appropriate frequency taken from ***Figure 1***. To project them the command, *soundsc(sec#, 11025)*, would be executed where *sec#* would be either *sec0, sec1,* or *sec2*. As for chords, they would be played within each duration block without any delay in between. To block out each audio section, the command, *pause()*, would be used and will last as long as the entirety of the duration length.

Without the loops, this was repeated for every time a note changed in pitch or frequency. If the duration had to modified, duration parameters had to rewritten and the line:

```
adsr = adsr_gen(Target,Gain,Duration);
```

Equation 3: *The adsr variable is the variable to encode the parameters of the entire note*

had to rewritten as well. By taking this process and repeating applying it to different notes with different lengths(pitches), “Africa” by Toto was able to be created.

Results

As result of this process and analysis, the outcome presents a heavily repeated amount of code that plays “Africa” from Toto using frequency sampled at 11025 Hz that was adjusted from 120 BPM to 93 BPM. The result is clean and has no problems while running the code. The audio projection of “Africa” will play and play uninterrupted till its completion. As described, only the treble clef () (right hand) part was coded. The left hand and the flute could have also been employed, but to simplify the execution of playing the file and meet the deadline, the only that part was converted was the treble clef (), as it is the most recognisable when the song is naturally played. To include the left hand and the flute, the individual frequencies will have to be translated from the sheet music as they run both on separate pitches and scales.

The only concern is that personally, on my laptop, loops could not be applied or properly tested as elaborated in ***Figure 5***. This makes re-editing slightly difficult as every single line was physically written out to capture that note, chord, or rest. If programmed on a stronger computer with a stronger processor, the chances to implement loops and optimize the code would be possible.

Conclusion

Overall, the project to recreate the C major scale, which turned into recreating the beginning 14 measures to “Africa” by Toto, was a constructive project in understanding DSP. It allowed for a better understanding of the ADSR concept, sampling frequency, and of debugging MATLAB code. The process of learning how to program this code also required the external skills of musical composition to read sheet of music and translate that back to various frequencies of varying octaves and specific scales. To properly attain the final product required patience and precision as unforeseen problems occurred that was beyond the control of MATLAB (***Figure 5***). This eventually could work itself into the entire recreation of “Africa”, but due to its structure without loops, it is quite inefficient to do so.

Appendix

Section 1

MATLAB function Code for adsr_gen.m:

```
function a = adsr_gen(target,gain,duration)
% Input
% target - vector of attack, sustain, release target values
% gain - vector of attack, sustain, release gain values
% duration - vector of attack, sustain, release durations in ms
% Output
% a - vector of adsr envelope values
fs = 11025;
a = zeros(fs,1); % assume 1 second duration ADSR envelope
duration = round(duration./1000.*fs); % envelope duration in samp

% Attack phase
start = 2;
stop = duration(1);

for n = [start:stop]
a(n) = target(1)*gain(1) + (1.0 - gain(1))*a(n-1);
end

% Sustain phase
start = stop + 1;
stop = start + duration(2);
for n = [start:stop]
a(n) = target(2)*gain(2) + (1.0 - gain(2))*a(n-1);
end
```

```
% Release phase
start = stop + 1;
stop = fs;
for n = [start:stop]
a(n) = target(3)*gain(3) + (1.0 - gain(3))*a(n-1);
end
```

<u>Section 2</u>

MATLAB Code for 14 measures of Toto's Africa

See m-file attached, if attached to this section, it will add the total page count by an extra 44 pages using size 12-font on Times New Roman

<u>Section 3</u>

"Pinkppermint10". (1982). Africa. [Recorded by Toto]. *Toto IV*. [PDF, MP3, & MIDI File]. Moorhead, Minnesota: muscore.com. (2016). Retrieved from <https://musescore.com/user/124709/scores/2060096>.

Toto [Toto Toto]. (2013, May 22). *Toto - Africa (Official Music Video)* [Video File]. Retrieved from <https://www.youtube.com/watch?v=FTQbiNvZqaY>.