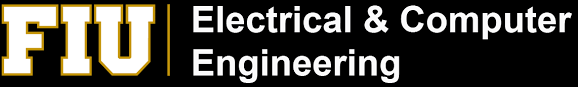
**4**

**ELECTRICAL AND COMPUTER ENGINEERING**

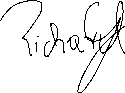
FIU College of Engineering and Computing

**EEL 3135: SIGNALS AND SYSTEMS**

|  |  |  |
| --- | --- | --- |
| Project Title | **Lab 04: Synthesis of Sinusoidal Signals—Music Synthesis** | |
| **Name:** | **Richard Hernandez** |  |
| Date | **12/4/19** | |

**Honor Pledge:**

"*On my honor, I have neither given nor received aid on this assignment.*"



|  |  |
| --- | --- |
| ***For Official Use Only*** | |
| ***Comments:*** | ***Grade / Score:*** |
| ***Graded by:*** |

1. OBJECTIVE

Music is a common example of signals, and a good way to practice the knowledge acquired about signals and systems in class. The main objective of this project is to learn and at the same time apply this formulas and systems into a real-life situation. By using signals with different frequencies and adding them up to acquire a new signal that will make an appealing sound when amplified by a system in this case the soundcard. The final result will be a MATLAB function that will synthesis a song by using the frequencies, amplitude, duration, other properties of the song.

1. WARM-UP and LAB EXERCISE

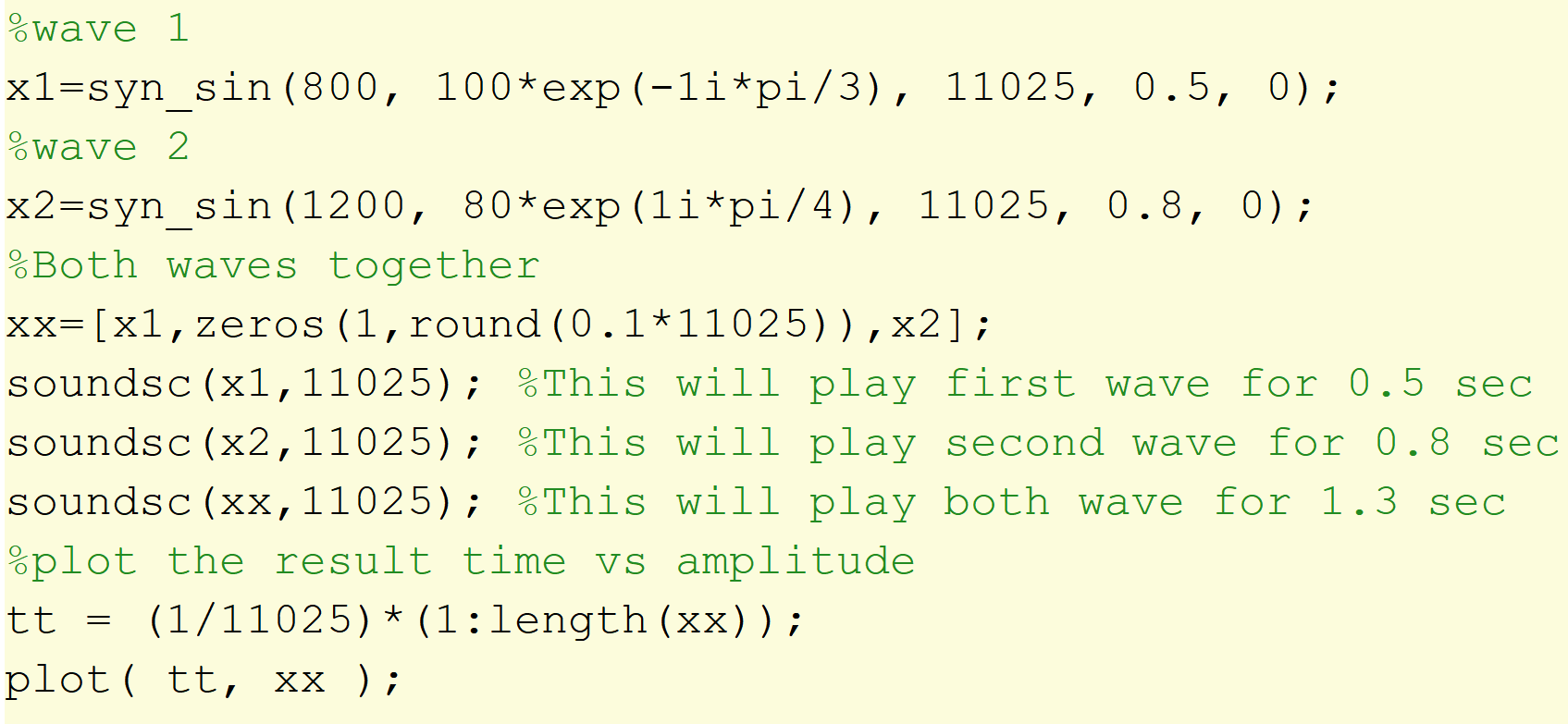
Section 2.2 **D-to-A Conversion:**

Because we are using a digital format we need to convert the signal of the music to analog, this will be done by the MATLAB function soundsc(). This function needs a vector and the advantage of using it is that the vector will be scaled automatically.

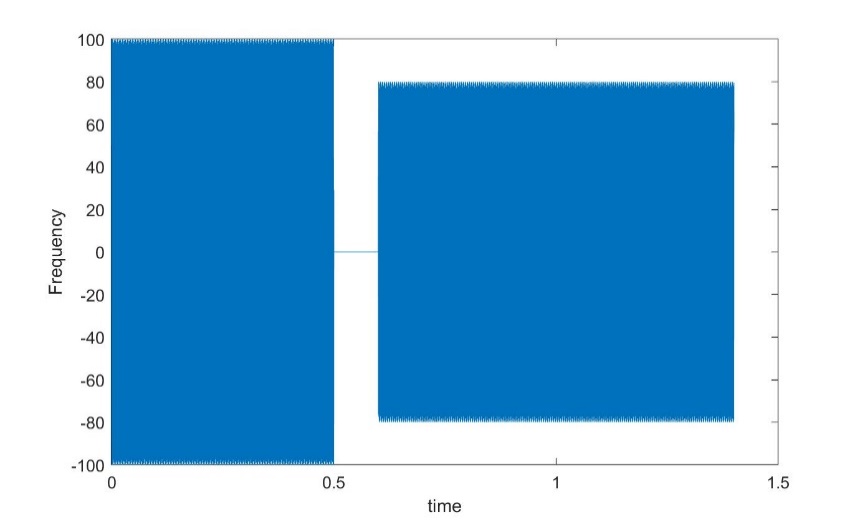
Now, during this section will use the function syn\_sin() which is given by Dr. Bai which will add synthesize cosine waves. This will create a wave, by using the parameters.

*Appendix: For the code of this function*

Next, task will be to create 2 waves using syn\_sin() called x1 and x2 and then unite them by concatenating them into a bigger vector called xx. The numbers used in the parameter were acquired from that section .



**Plot Result:**



Section 2.3 **Structures in MATLAB:**

A really smart way to represent the waves is by using MATLAB structures which can help us group the information of a wave in one variable. By using a dot after the variable we can add a property for example when a wave named “x”:

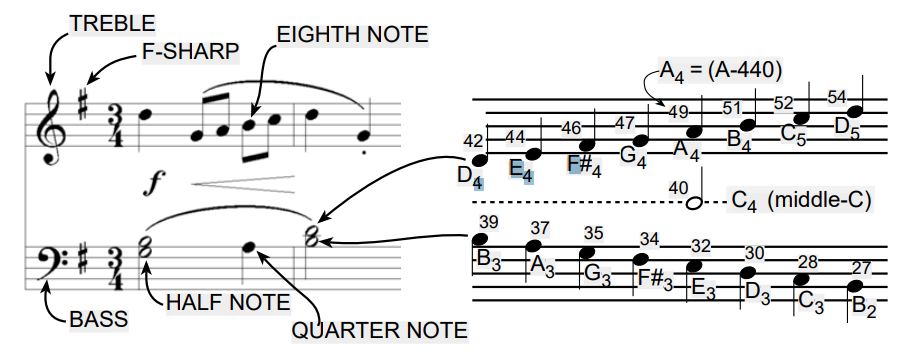
x.Amp = 5; --- Set amplitude of wave x to 5

x.phase = -pi/3 ---- set phase

This is just two examples how we can add more information inside the same variable. In addition, the types stored in each structure can be changed[vectors, strings, numbers, etc]

Section 2.5 **Piano Keyboard**:

This section was about how Pianos keys are divided and how they can be helpful to implement a better way to get the notes by using an pattern. By using this patterns we can algorithmically get the frequency of the note by just giving as an input the note number. Meaning that when the computer is given a number by using a function we can get the frequency, amplitude, duration and other valuable information.



We can also use the ratio 2^(1/12) to calculate any frequency of the notes anywhere on the keyboard. On the other hand, musical notation is a time-frequency diagram. And the shape of the notes defines the duration of the note.

Warm up

Section 3.1 **Note Frequency Function**:

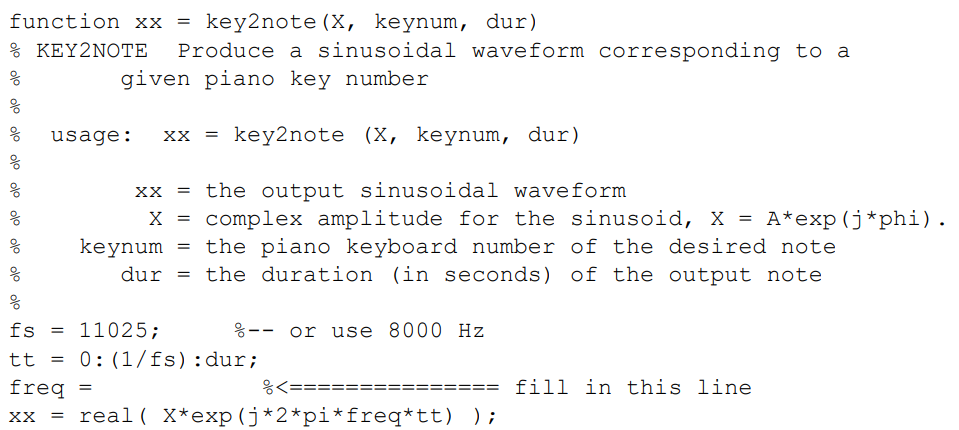
Creating a function to help in the creating of a waveform that will represent and sound as an specific key on the piano. The duration of this waveform will be given to the function as a parameter.

Function name: key2note

parameters: complex amplitude, number of the key, and duration of the note.

Returns a vector with the *real* part.

This is the code needed for this function but is missing the most important part since is missing the formula to calculate the frequency of the note.



By referencing the section before we can use the ratio and the key number to calculate the “*freq*” variable. This ratio will be multiplied by the 440 since that frequency of the A-440.

**freq = 440\*(2^((keynum-49)/12));**

Now when this formula is added we now have a function that returns a note vector every time called using the required key and parameters. This gets us one step closer to the synthesis of the song.

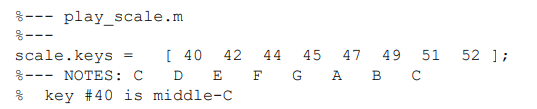
A close up of a mans face

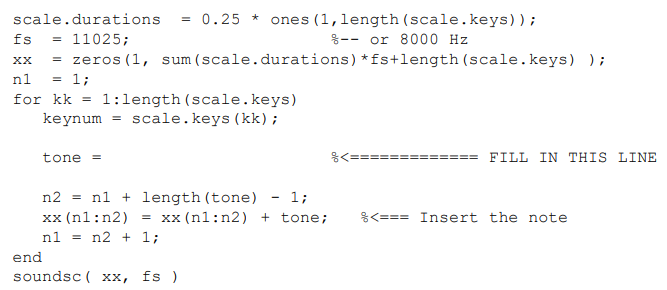
Description automatically generated

Section 3.2 **Synthesize a Scale:**

Scale will play a series of keys one after the other, meaning that the use of key2note() is required to get the frequency of each of the notes. The next function will be called play\_scales() and there is no return value, and no parameters either.

The following code is given showing us the keys that will be pressed:





Steps of Code:

1. The code above will apply a duration to each note of 0.25 sec(QUARTER).
2. The sampling rate will be 11025 since it create a smooth sound and is typically use. After that “*xx*” will be use to make vector of 0 to represent the time of the whole scale.
3. After this a loop will go through every key from the scales.key adding the note information to the “xx” which uses n1:n2 as index to plot tones in correct places.
4. Lastly play the sound saved in “xx”.

Using the function to get the note.(*Maybe be change later to implement more features*)

**tone= key2note(X, keynum, scale.durations(kk));**

Section 3.3 **Spectrogram: Two M-files:**

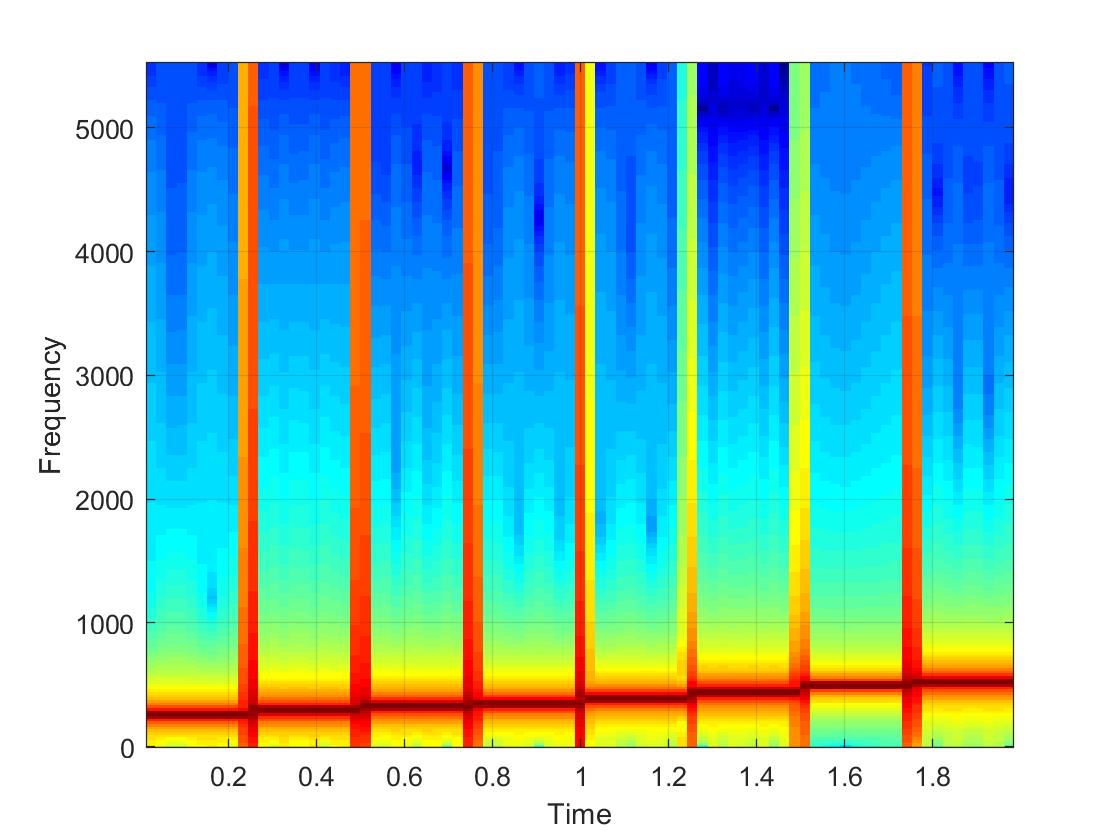
Part A:

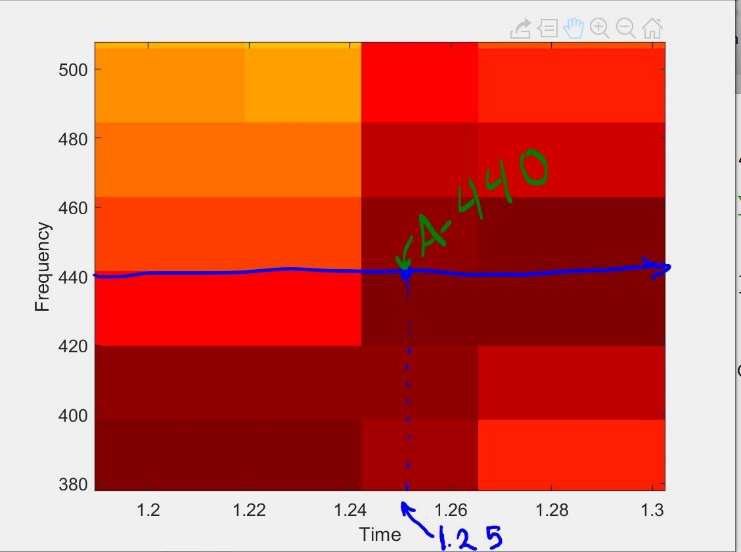
After the code added in section 3.2 the sound play one after the other.

Part B:

Now using the function specgram(xx,512,fs). The first parameter is the notes vector, second is the length of the window, and third is the sampling rate.

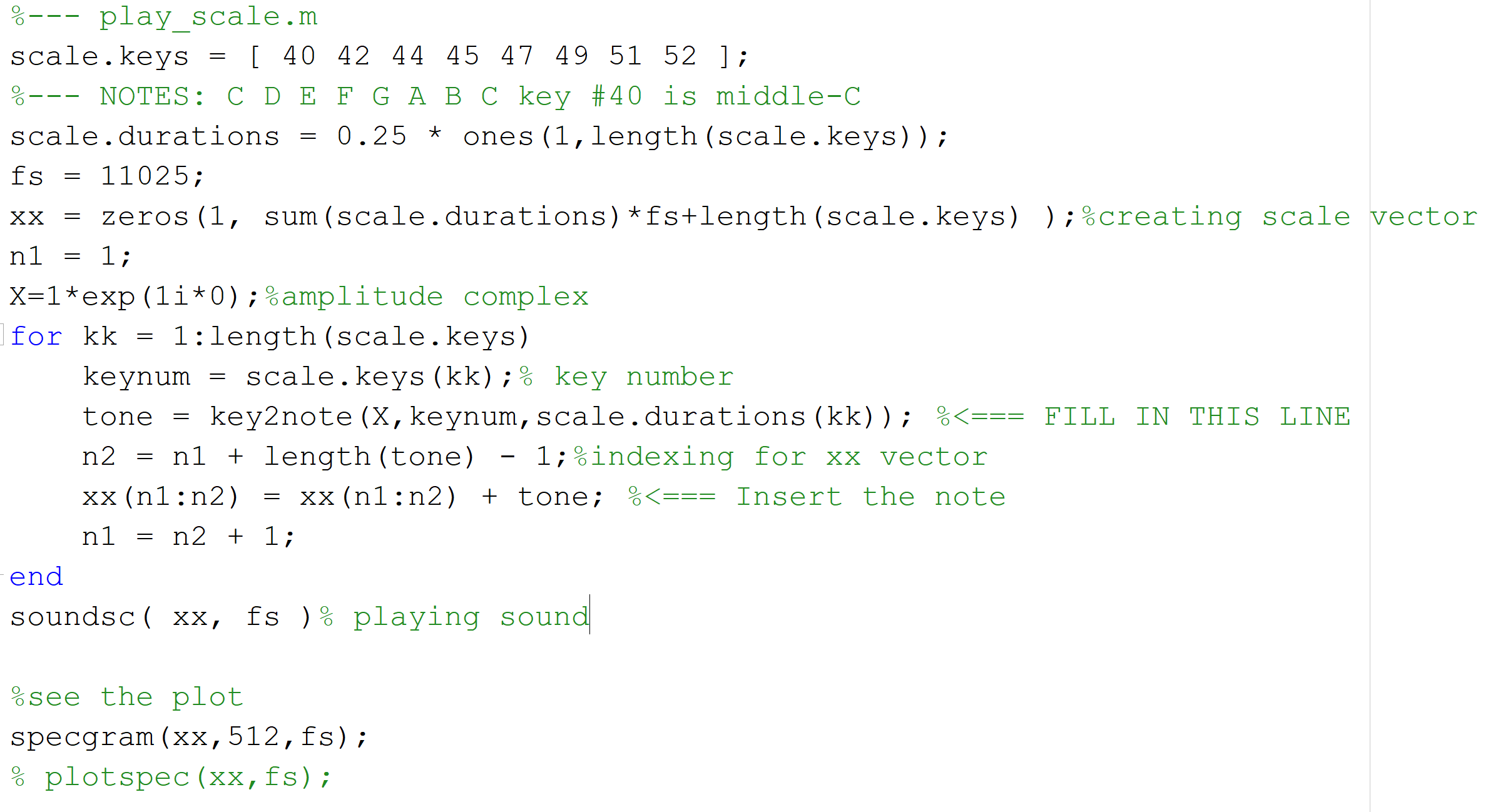
**Using specgram(xx,512,fs)**

Based on this image we can get information about what is happening.

* The time axis is 2 seconds long because we playing 8 notes for 0.25 seconds each.
* Also we can see the frequency increasing gradually each 0.25 sec.
* Between each note there is some kind of cut-off (this can be heard when the notes change).
* Here we can see a closer look at the frequency 440 Hz which is key number 49.
* The time of switch was 1.25 sec.

*Plotspec(xx,fs) is not currently installed in my version of MATLAB.*

Code:



*The code in this picture is the one used to play the scale and generate the specgram graph shown in this section.*

**Section 4 Lab Exercise: Synthesis of Musical Notes**

We the knowledge acquired on the section before constructing a script that will play all the notes in a MATLAB structure loaded from bach\_fugue.mat.

1. First we need to create this script
2. Play the song
3. Add envelop and harmonics to improve the sound of the song.

Part A:

The sampling frequency used on the synthesis of the songs will be 11025 since the computer supports this amount of sampling thanks to the amount of memory that it contains.

fs = 11025;

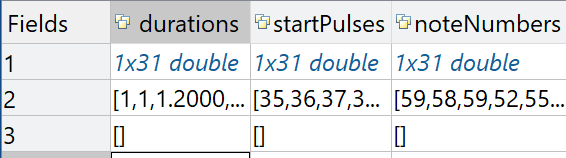
Part B

The sampling rate is now set, we can load song by using the function load from MATLAB, this will add a structure to the environment called theVoices.

Inside this structure there is 3 fields:

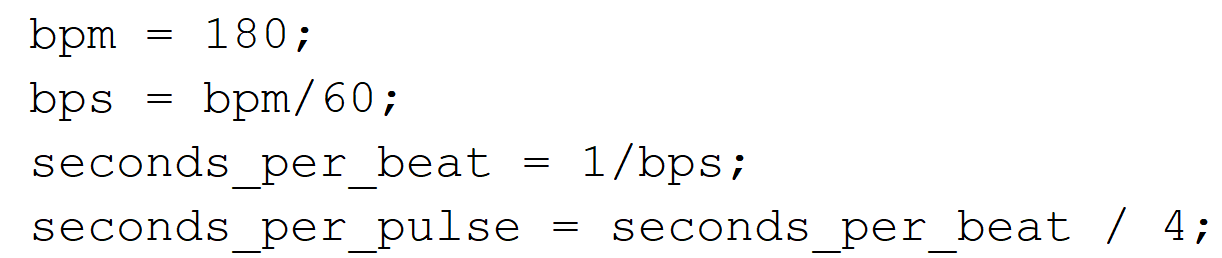
1. **noteNumbers**: number of the key used on the key2note function for frequency
2. **startPulses**: it indicates at what pulse will the note begin.
3. **Duration**: how long will the note will be sounding(duration of pulse)

Shown here:



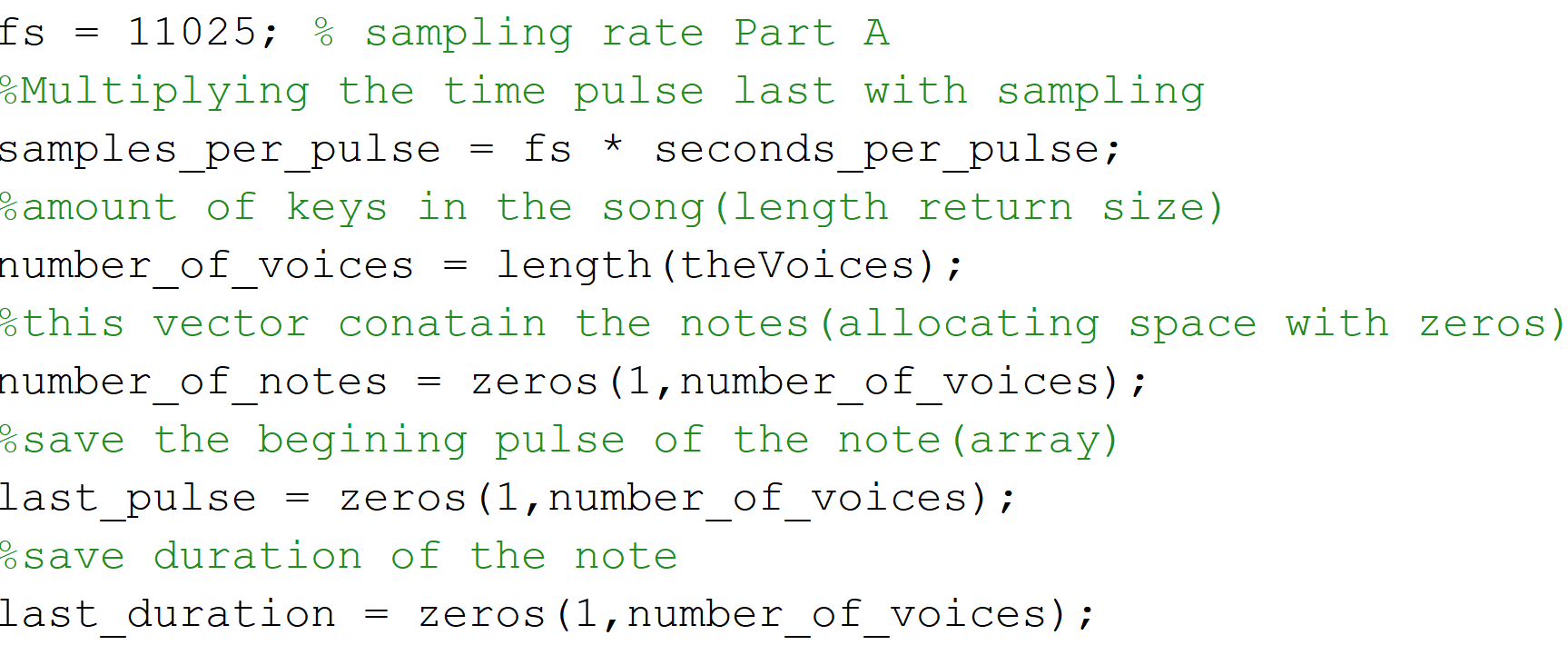
**Pulses:** is based on the measures and beats which are the most basic time intervals in music score.

To calculate the pulse beats per minute is the prefer term by musicians, which needs to be divided by 60 seconds to get the beats per second. Now we are able to obtain the amount of time that a beat has by dividing 1/ “beats per second”. Therefore, we can calculate the amount of time that the pulse will last by dividing it by 4. The code that follows will show how this is programmed in MATLAB.

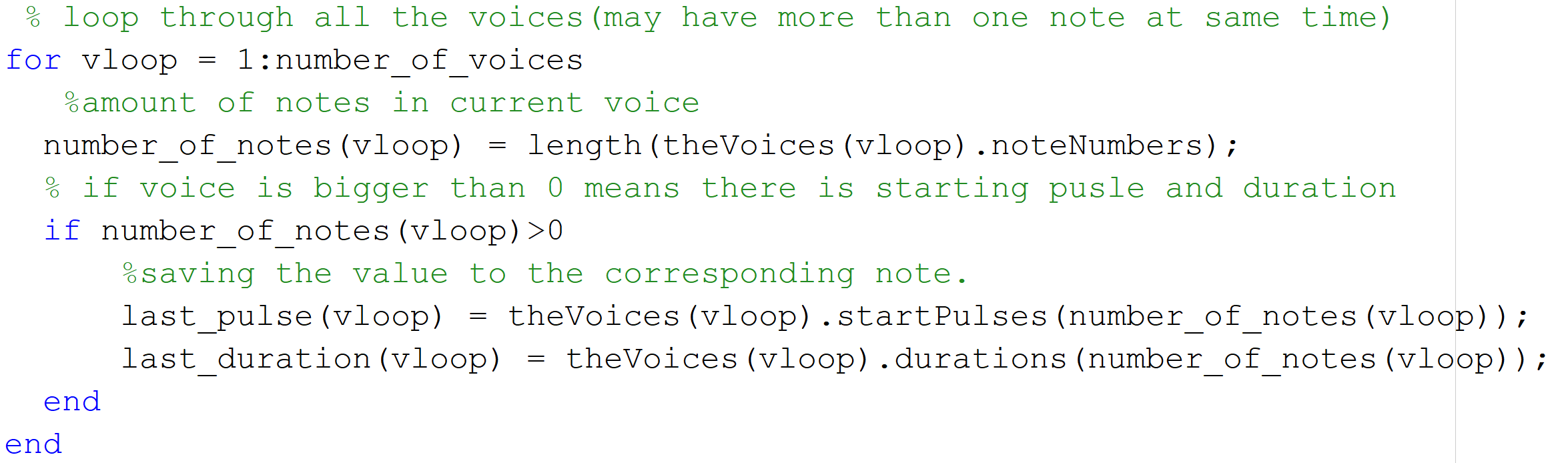


Notes: incrementing the bpm will play the song faster because seconds per beat is will be smaller.

The code that follows will set up multiple vectors or arrays to that will match with the taming of the voices. This will help because some notes will play for a longer time than other so is good idea to keep track of their timing individually.



To obtain all the information from theVoice structure a loop will be needed, so that every single voice is stored on the variables above. On the other hand timing can be issue since some voices contain different notes inside, and not every note last the same amount of time causing some of them to overlap. This loop will set get the notes per voice(maybe more than 1) and get their time and save it as well by using vloop as an index.

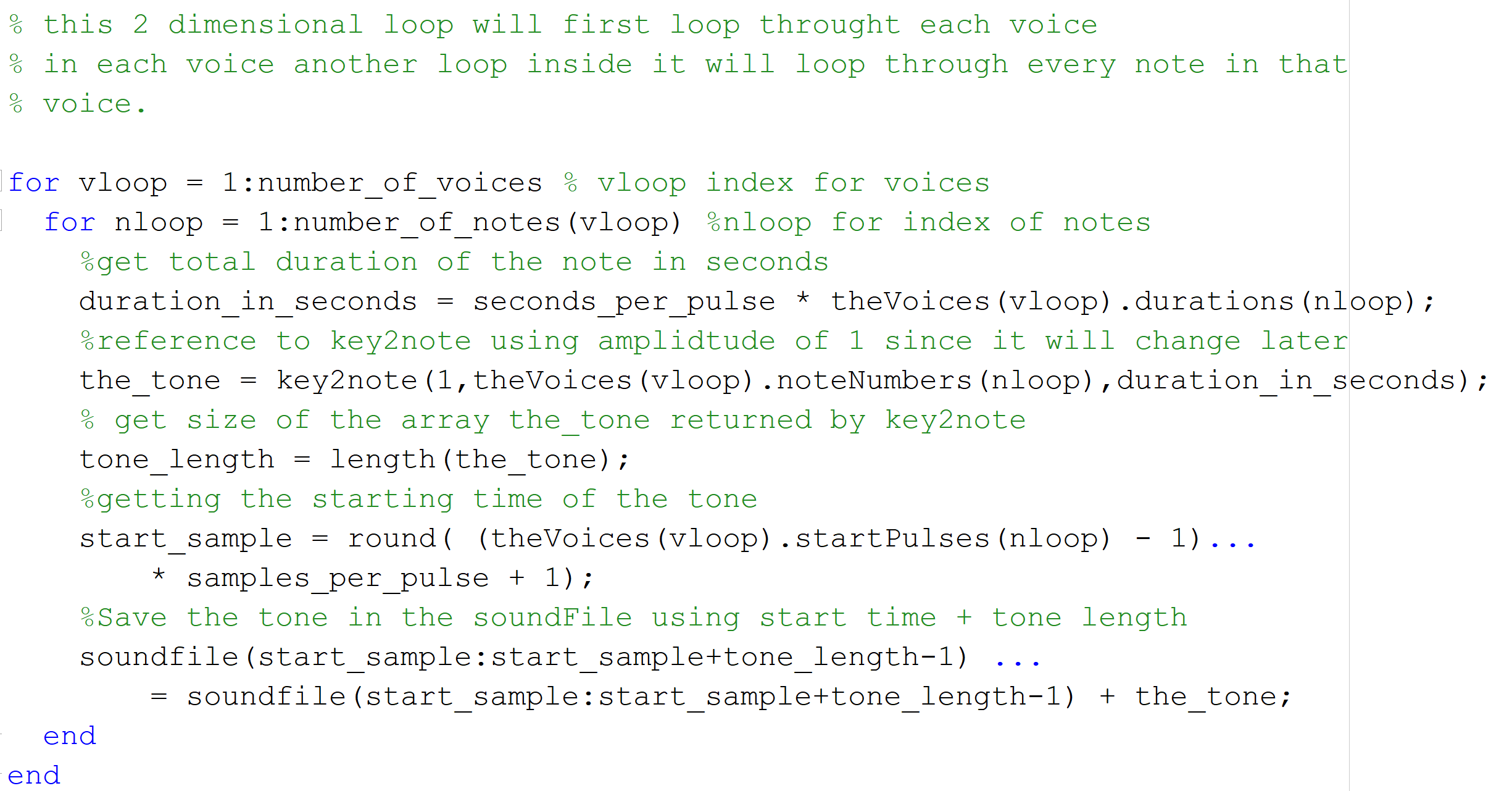


A vector which will contain all of the notes together with the amplitude and pulse information will be saved as soundFile. This snippet of code will allocate the size of the variable by getting the largest starting pulse and largest duration, adding them. After that multiply samples\_per\_pulse which will give the value of the largest amount of time then ceil it. Which mean to round it up.

soundfile = zeros(1, ceil( samples\_per\_pulse \* (max(last\_pulse) + max(last\_duration)) ) );

The variables needed to play the song are ready now is time to get the frequency of each note and save the information on the soundfile vector.

1. Get the duration of the note in seconds.
2. Get the frequencies from key2note function
3. Calculate the length of the tone
4. Calculate the starting time of the note
5. Add it to the sound file, in the correct time.



Part C

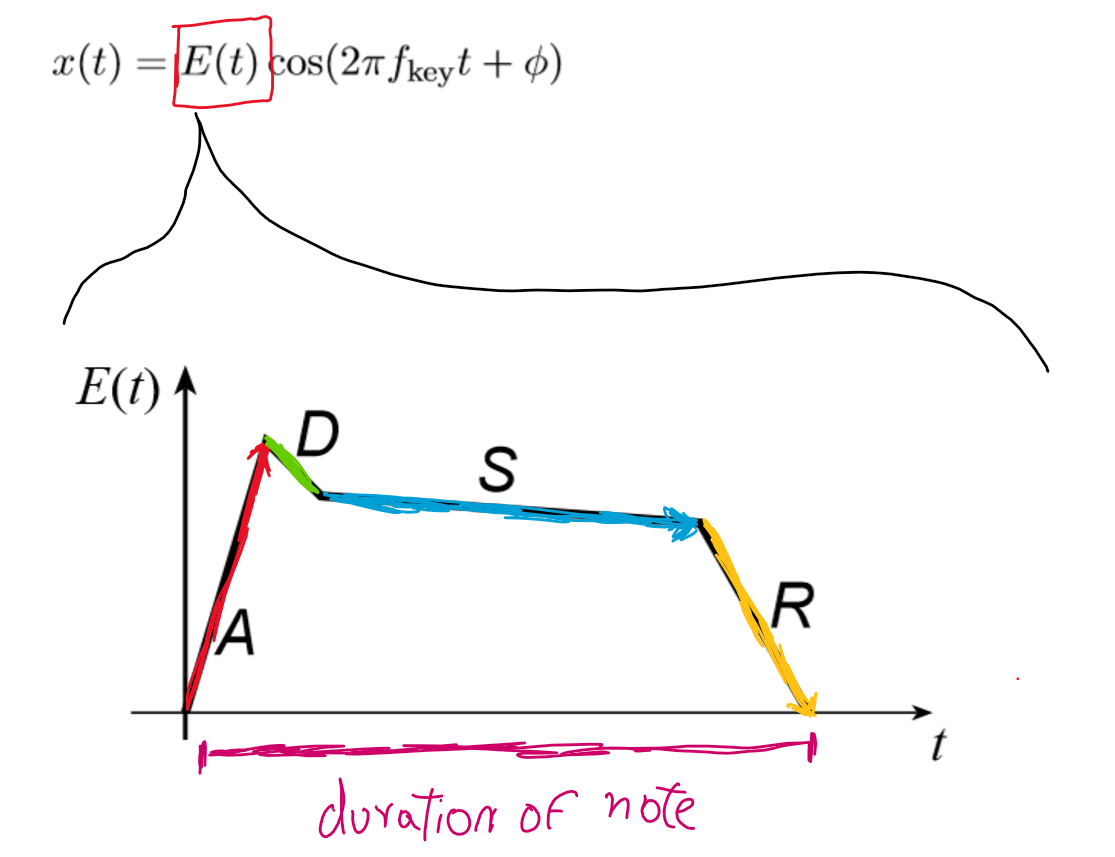
Now synthesize the created waveform saved in soundFile, by using the function soundsc(). The sound can vary depending on the time set on the bpm variable.

*Song bach\_fugue.mat*

* When set to 180 bpm I found the song to be too fast, not been able to distinguish many notes. While testing for some other speed 140 bpm will sound really smooth and you can nicely distinguish the changes on the notes.
* There still some harsh sounds in the beginning of each notes. On part D will show the spectrum without ADSR.

**Adding ADSR(Envelope):**

Now to explain how this will change the song, I made emphasis on a weird sound at the beginning of each note, this is because the frequency is immediately set high and then abruptly turn down. Creating harsh sounds, when changing the notes. So envelope will be multiplied by the frequency to create “fade in” and “fade out” effect.



E(t) is the value for which the function will increase or decrease.

A - fade in

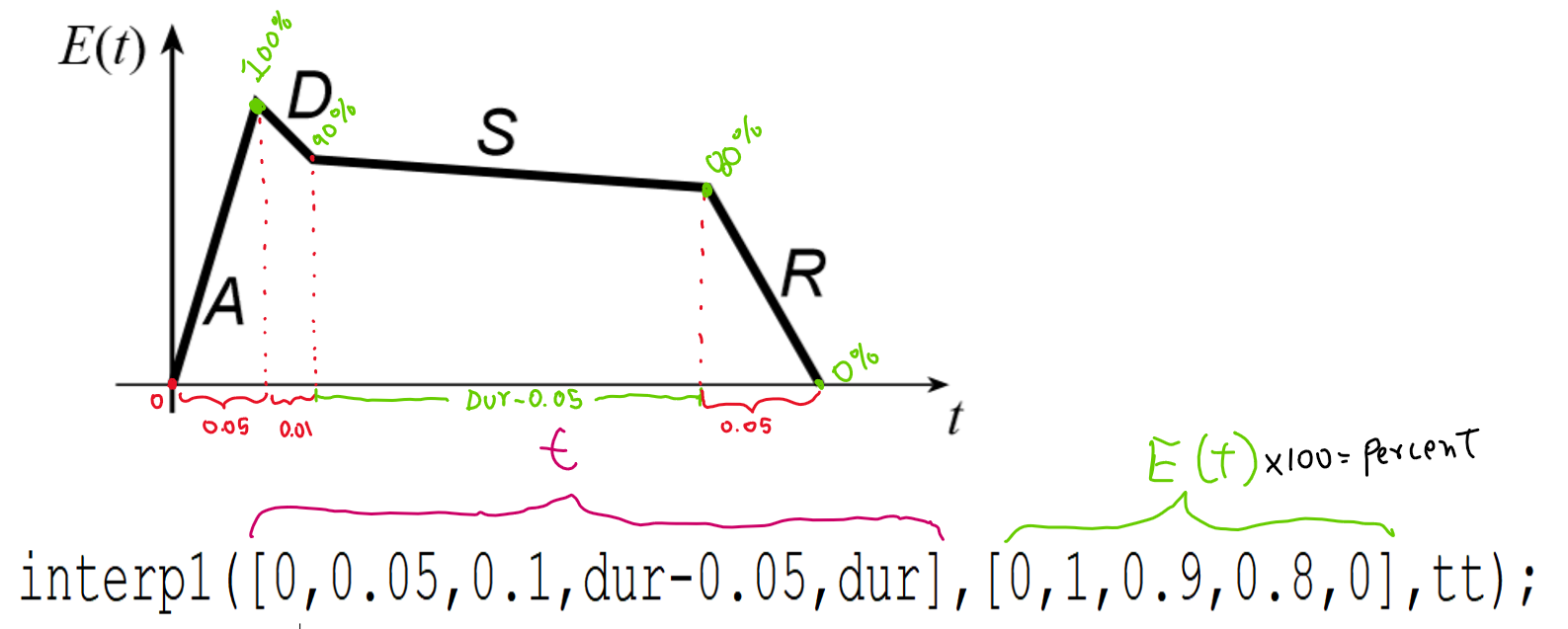
D – delay (short duration drops)

S – Sustain(gradually decrease but not so much, need to seem constant)

R – Release(quickly go down to zero)

The effect of this change will dramatically change the song for the best. Creating a really nice sound since the notes don’t suddenly stop. In theory this how it works but to program it will use a function called interp1().

Interp1: create a line of point from one point to another by using interpolation. The parameter needed for this function are 2 arrays, and a vector shown here:



The image above uses an array of key point duration and create a line of points growing up to the percentage specified on the second array. This the normal case for any note that last more than 0.15 sec. In the adsr script that follows there is two more cases since notes that are too short are barely noticeable, so we use different percentages.

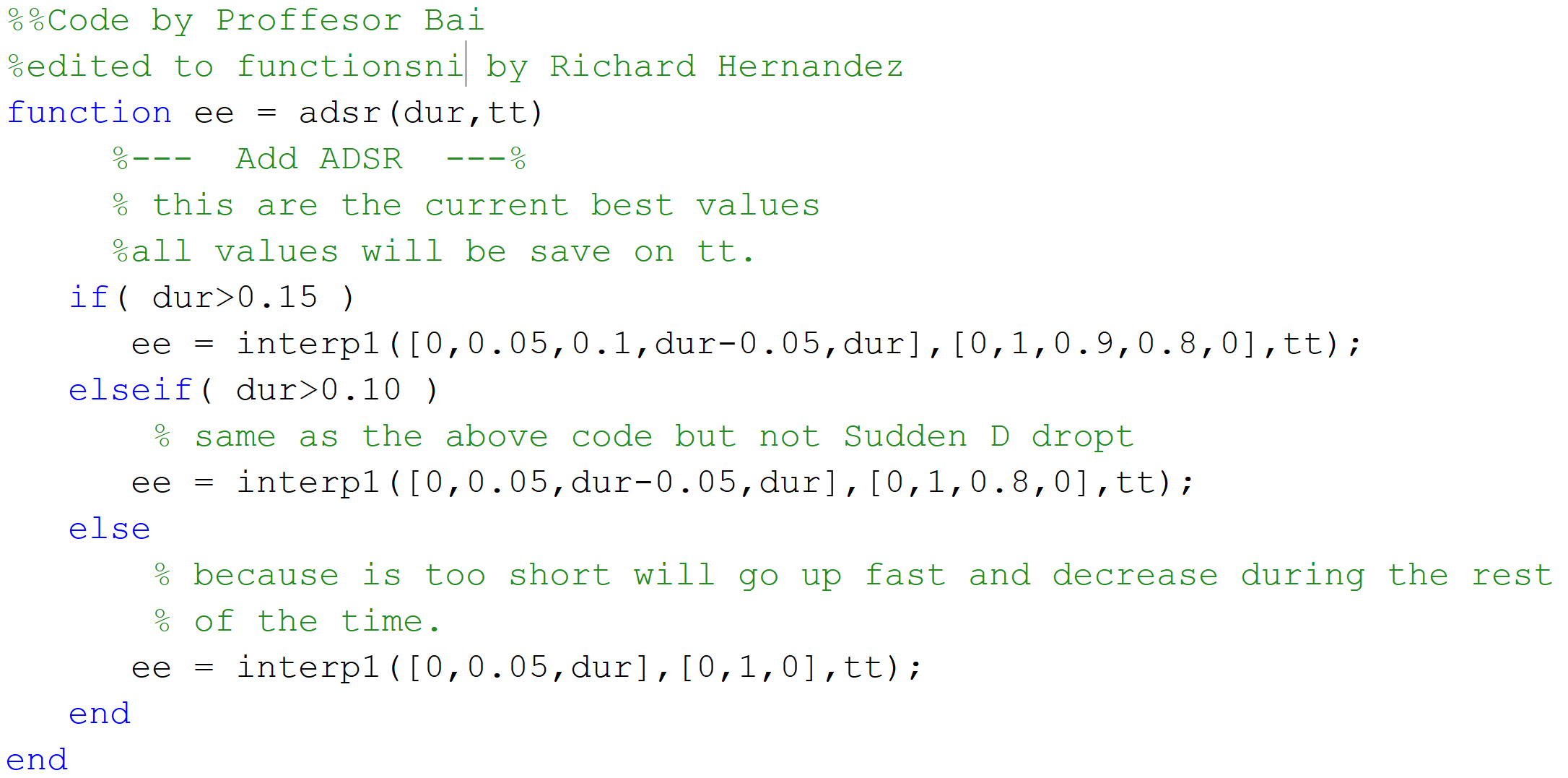
In the example above :

A - will increase from 0 to 100% of the amplitude in 0.05(or 5% of the time)

D – will drop from 100% to 90% of the amplitude in 0.01(1 % of the time)

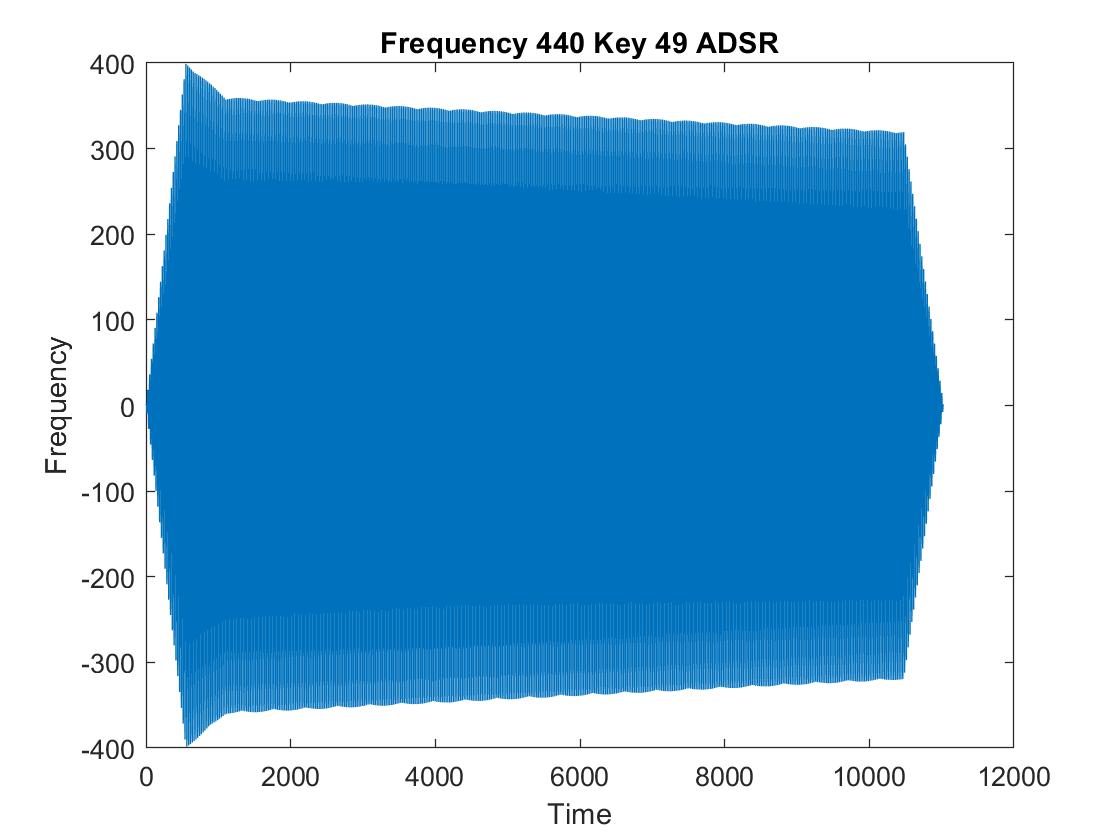
S – between 90% dropping to 80% during most of the time of the duration.

R – will drop to 0% in the last 5% of the time.

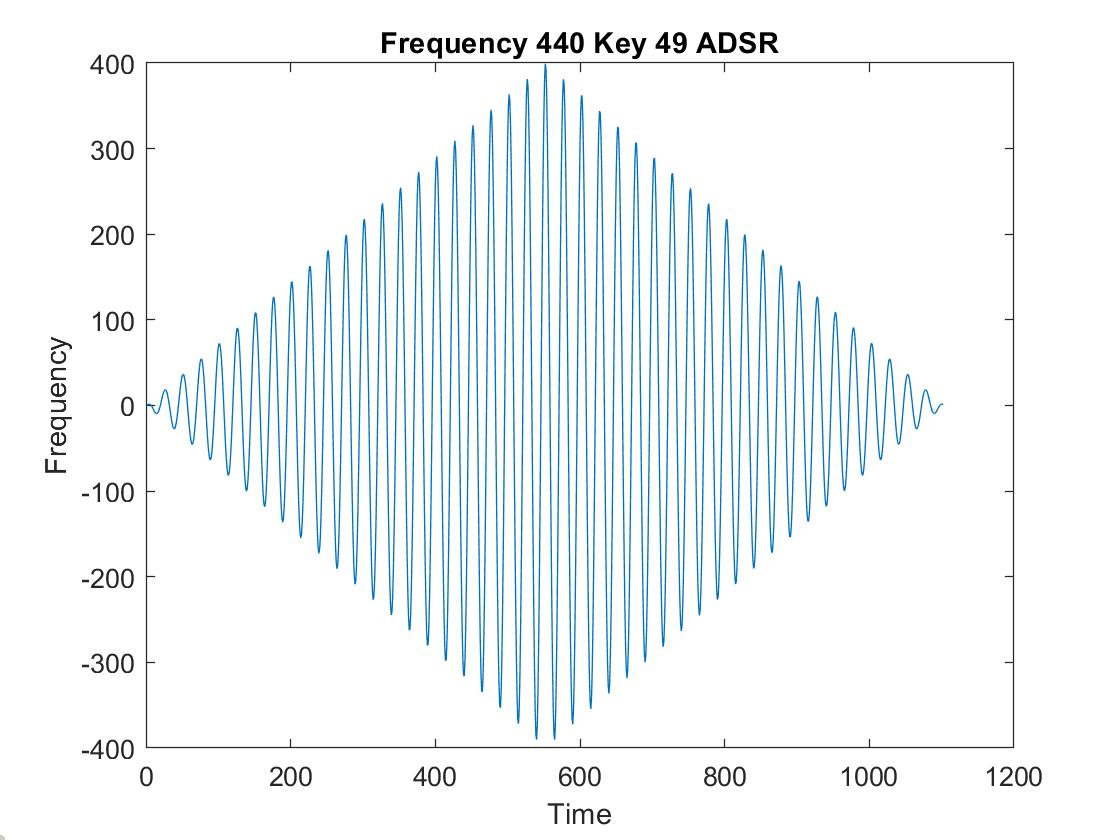


Note: *The vector returned will be multiplied by the note vector of key2notes.*

**The duration of this example is 1 second.**

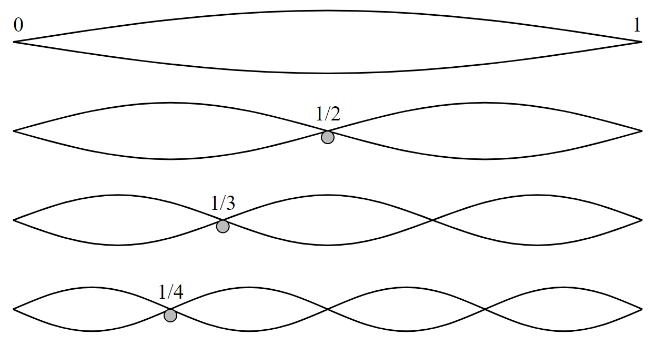
**

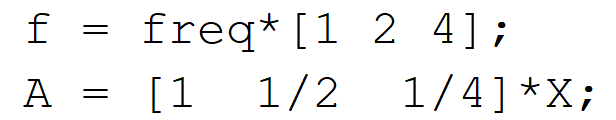
**The duration here is 0.10 seconds making it too small to apply regular ADSR.**

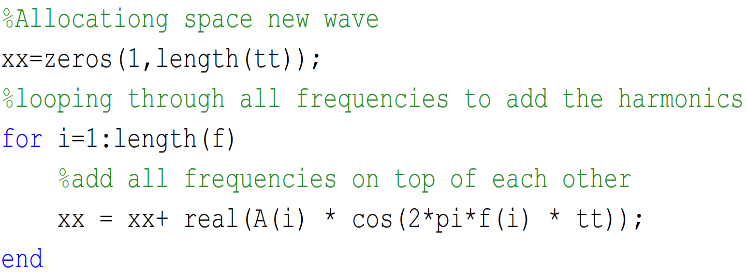


**Add Harmonics:**

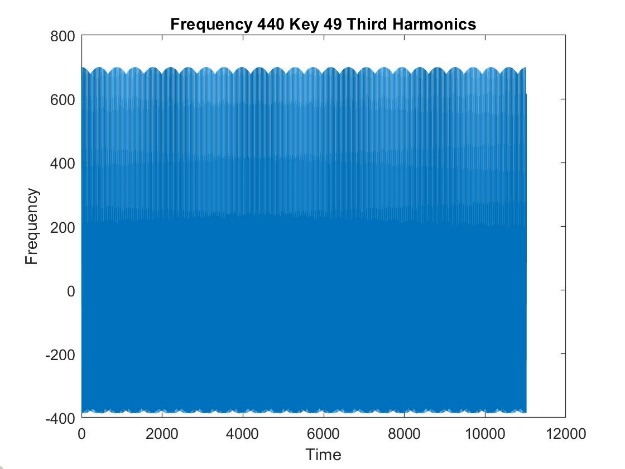
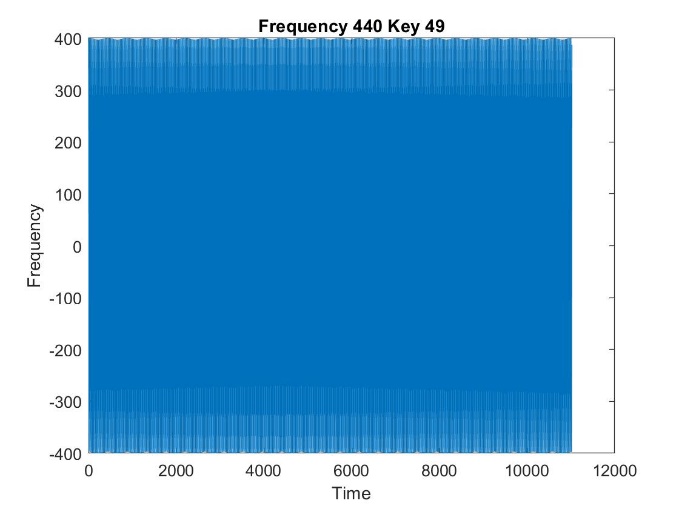
Harmonics are really difficult to perceive, nevertheless they are present in most song, harmonics are positive integers that are multiplied with the fundamental frequency. Also, they higher you take your harmonic the waves on the frequency will be more noticeable.

An example of this is imagine you got a frequency of 100 Hz if you multiply it by 2 is 200 Hz, by 3 is 300 Hz. The fundamental frequency is called first harmonics, the 200 Hz second one, and third would be 300 Hz. Of course, the frequency increases and the amplitude (volume) will be divided by that number too.

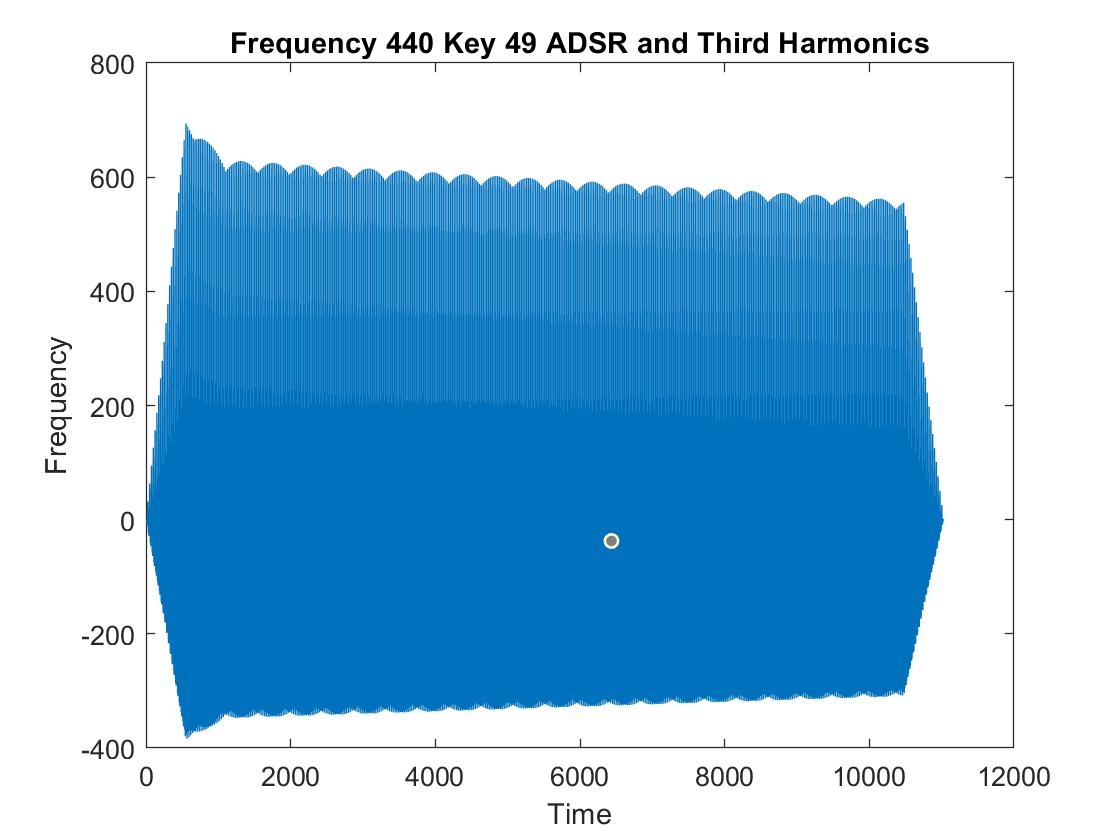
So in our code of multiply the fundamental the key currently been converted, the higher you go the higher the pitch of the note. The amplitude will be inversely decreased as frequency goes higher.



Now we add the frequencies on top each other. In the pictures below you can see that the frequencies of same note incremented by more than 200 Hz



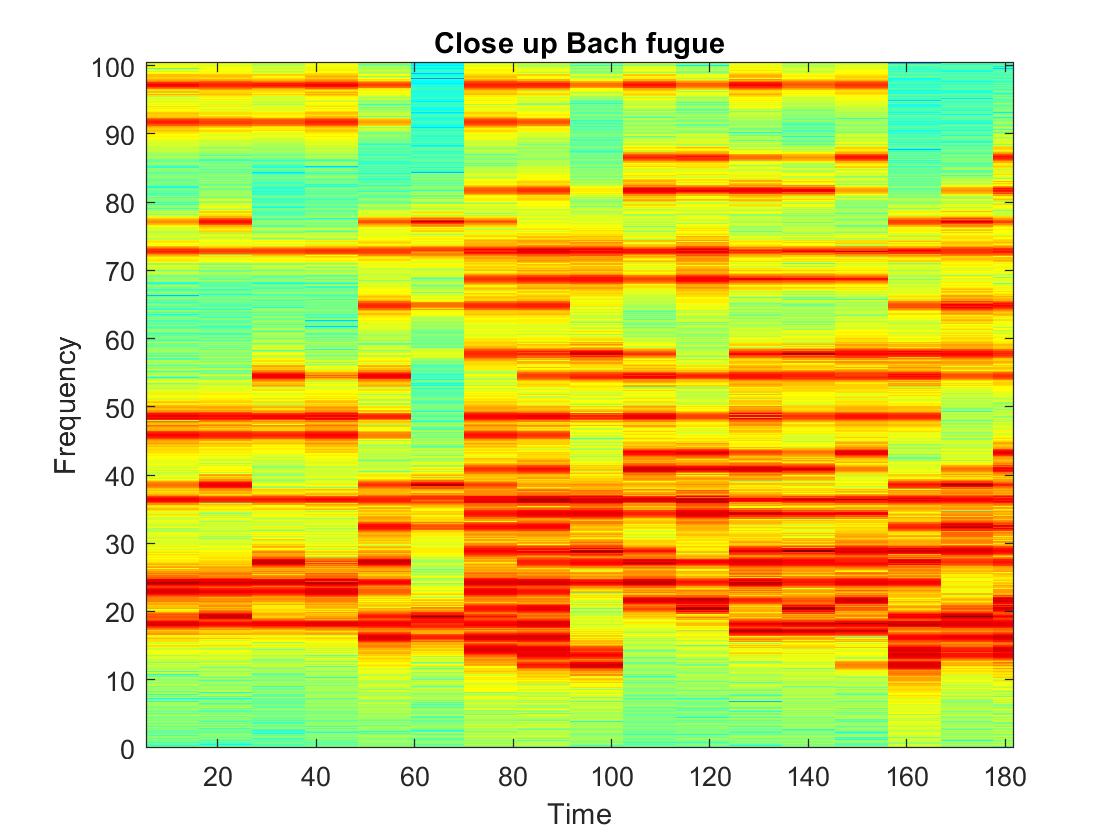
The current look of a keynote using ADSR and Harmonics:



Part D

A close up of a logo

Description automatically generated

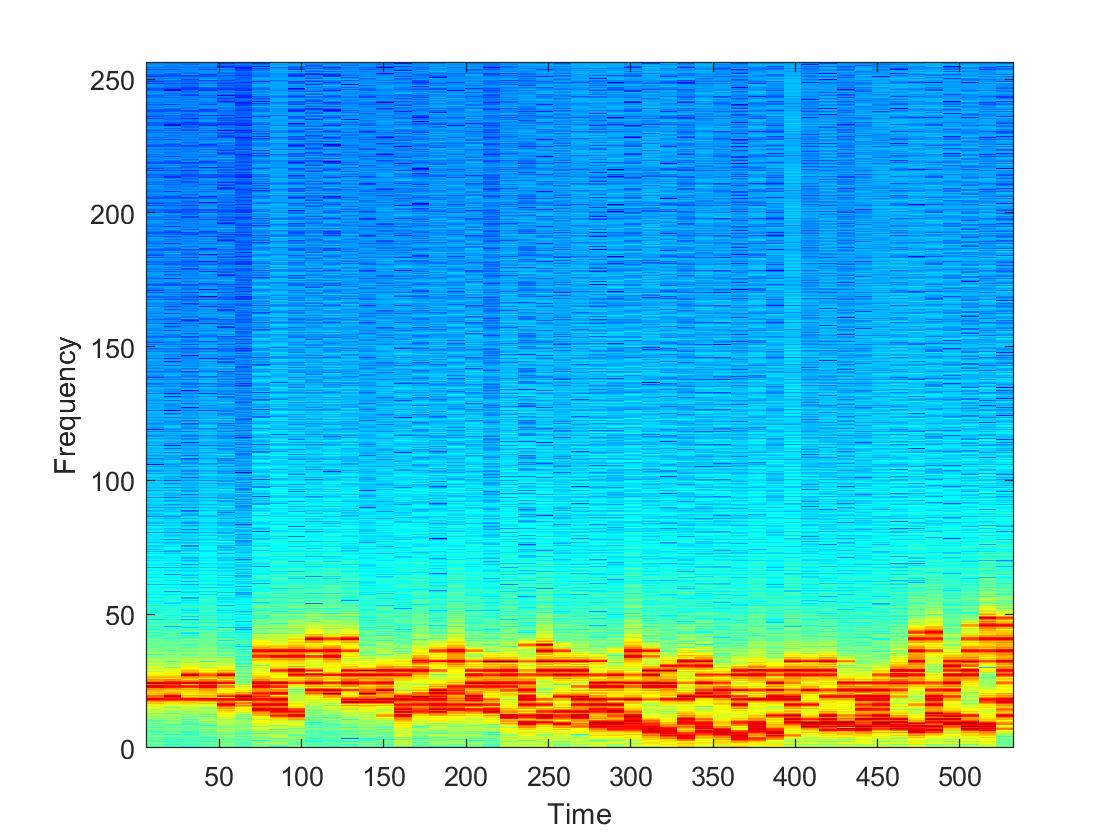


***No ADSR or harmonics just regular Fundamental Frequencies.***

*A close up of a piece of paper

Description automatically generated*

***Only ADSR applied no Harmonics***

**

1. DISCUSSION

* *Describe your results and compare them to expected results or the theoretical values obtained through the calculations.*
* *Discuss some issues that you faced and how you overcame problems and made your code better*

1. APPENDIX

Section 2.2: **syn\_sin() by Dr Bai:**

function [xx,tt] = syn\_sin(fk, Xk, fs, dur, tstart)

%SYN\_SIN Function to synthesize a sum of cosine waves

% usage:

% [xx,tt] = syn\_sin(fk, Xk, fs, dur, tstart)

% fk = vector of frequencies

% (these could be negative or positive)

% Xk = vector of complex amplitudes: Amp\*eˆ(j\*phase)

% fs = the number of samples per second for the time axis

% dur = total time duration of the signal

% tstart = starting time (default is zero, if you make this input optional)

% xx = vector of sinusoidal values

% tt = vector of times, for the time axis

%

% Note: fk and Xk must be the same length.

% Xk(1) corresponds to frequency fk(1),

% Xk(2) corresponds to frequency fk(2), etc.

if nargin<5, tstart=0; end %--default value is zero

if length(fk)~=length(Xk), error('The vector length of freqeuncies and complex amplitudes should be the same!'); end

if ~isscalar(fs), error('The signal sampling rate ''fs'' should be a scalar!'); end

if ~isscalar(dur), error('The time duration of the signal ''dur'' should be a scalar!'); end

if ~isscalar(tstart), error('The starting time of the signal ''tstart'' should be a scalar!'); end

tt=tstart+0:1/fs:dur;

xx=zeros(1,length(tt));

for i=1:length(fk)

xx=xx+real(Xk(i)\*exp(1i\*2\*pi\*fk(i)\*tt));

end

%--- **play\_scale.m**

scale.keys = [ 40 42 44 45 47 49 51 52 ];

%--- NOTES: C D E F G A B C key #40 is middle-C

scale.durations = 1 \* ones(1,length(scale.keys));

fs = 11025;

xx = zeros(1, sum(scale.durations)\*fs+length(scale.keys) );%creating scale vector

n1 = 1;

X=1\*exp(1i\*0);%amplitude complex

for kk = 1:length(scale.keys)

keynum = scale.keys(kk);% key number

tone = key2note(keynum,fs,scale.durations(kk),1); %<=== FILL IN THIS LINE

n2 = n1 + length(tone) - 1;%indexing for xx vector

xx(n1:n2) = xx(n1:n2) + tone; %<=== Insert the note

n1 = n2 + 1;

end

soundsc( xx, fs )% playing sound

%see the plot

specgram(xx,512,fs);

% plotspec(xx,fs);

Key2Note.mat

function xx = key2note(X,keynum,dur)

%KEY2NOTE Synthesize sinusoids from key number and duration.

% x = KEY2NOTE(keynum,dur,fs) creates the sinusoid for a

% particular key number and duration (in seconds). fs is the

% base sampling frequency of the output device.

% Original code by James H. McClellan

% Rev. Dr. Bai, 01/10/2013

fs = 11025; %-- or use 8000 Hz

tt = 0 : 1/fs : dur-1/fs;

if keynum==0 % Silence

xx = zeros(size(tt));

else

freq = 440\*(2^((keynum-49)/12));

% %% Without Harmonics

% xx = real( X\*exp(1i\*2\*pi\*freq\*tt) );

%% With Harmonics

f = freq\*[1 2 4]; % Add harmonic components

A = [1 1/2 1/4]\*X; % Amplitudes for harmonic frequencies

%Allocationg space new wave

xx=zeros(1,length(tt));

%looping through all frequencies to add the harmonics

for i=1:length(f)

%add all frequencies on top of each other

xx = xx+ real(A(i) \* cos(2\*pi\*f(i) \* tt));

end

% --- Add ADSR ---%

ee = adsr(dur,tt);

xx = xx.\*ee;

end

syn\_song.mat

bpm = 140;

bps = bpm/60;

seconds\_per\_beat = 1/bps;

seconds\_per\_pulse = seconds\_per\_beat / 4;

fs = 11025; % sampling rate Part A

%Multiplying the time pulse last with sampling

samples\_per\_pulse = fs \* seconds\_per\_pulse;

%amount of keys in the song(length return size)

number\_of\_voices = length(theVoices);

%this vector conatain the notes(allocating space with zeros)

number\_of\_notes = zeros(1,number\_of\_voices);

%save the begining pulse of the note(array)

last\_pulse = zeros(1,number\_of\_voices);

%save duration of the note

last\_duration = zeros(1,number\_of\_voices);

%% The voices might not be finished together.

% loop through all the voices(may have more than one note at same time)

for vloop = 1:number\_of\_voices

%amount of notes in current voice

number\_of\_notes(vloop) = length(theVoices(vloop).noteNumbers);

% if voice is bigger than 0 means there is starting pusle and duration

if number\_of\_notes(vloop)>0

%saving the value to the corresponding note.

last\_pulse(vloop) = theVoices(vloop).startPulses(number\_of\_notes(vloop));

last\_duration(vloop) = theVoices(vloop).durations(number\_of\_notes(vloop));

end

end

%% Fir for the last voice finished

soundfile = zeros(1, ...

ceil( samples\_per\_pulse \* (max(last\_pulse) + max(last\_duration)) ) );

% this 2 dimensional loop will first loop throught each voice

% in each voice another loop inside it will loop through every note in that

% voice.

for vloop = 1:number\_of\_voices % vloop index for voices

for nloop = 1:number\_of\_notes(vloop) %nloop for index of notes

%get total duration of the note in seconds

duration\_in\_seconds = seconds\_per\_pulse \* theVoices(vloop).durations(nloop);

%reference to key2note using amplidtude of 1 since it will change later

the\_tone = key2note(1,theVoices(vloop).noteNumbers(nloop),duration\_in\_seconds);

% get size of the array the\_tone returned by key2note

tone\_length = length(the\_tone);

%getting the starting time of the tone

start\_sample = round( (theVoices(vloop).startPulses(nloop) - 1)...

\* samples\_per\_pulse + 1);

%Save the tone in the soundFile using start time + tone length

soundfile(start\_sample:start\_sample+tone\_length-1) ...

= soundfile(start\_sample:start\_sample+tone\_length-1) + the\_tone;

end

end

specgram(soundfile,fs,512);

soundsc( soundfile, fs )

1. SELF VERIFICATION SHEET

* *If the section (or a part) is mainly done by Student A, Student B will verify the result and sign the sheet; and vice versa.*