PROJECT REPORT

May 4, 2020

Hamad El Kahza
New York Institute Of Technology
Department of Electrical and Computer Engineering

New York Tech

Making Music with MATLAB

May 4, 2020

1 Introduction

This project intends to create sound waves using Matlab at an audio frequency ranging between (300 Hz <= f <= 3 kHz). The specific goal is to generate music evaluating the following function at equal time intervals.

$$v = Asin(2\pi f t) \tag{1.1}$$

As a kid, I always dreamed about having arcade machines at home so I wouldn't have to spend loads of money at the arcade to figure out a way to beat a friend in Mortal Kombat. The theme song of this game is a nostalgic memory. MATLAB has the ability to output a time series to the sound card of a computer and this was a great opportunity to tweak the engineering and recreate the theme song I grew up enjoying.

1.1 Generating the song

To generate the desired audio output, I started by retrieving the necessary fundamental notes to recreate a sample of the song. For this purpose, I used the American Standard Pitches as instructed, as well as a combination of other notes with custom pitches. Additionally, some notes are built by summing up to additional sin waves that need to be played at the same time. For instance:

$$A\#A = (A\# + A) = sin(2\pi 220t) + sin(2\pi 440t); \tag{1.2}$$

The complete list of the notes utilized can be found in the code in Appendix A

1.2 Procedure and Objectives

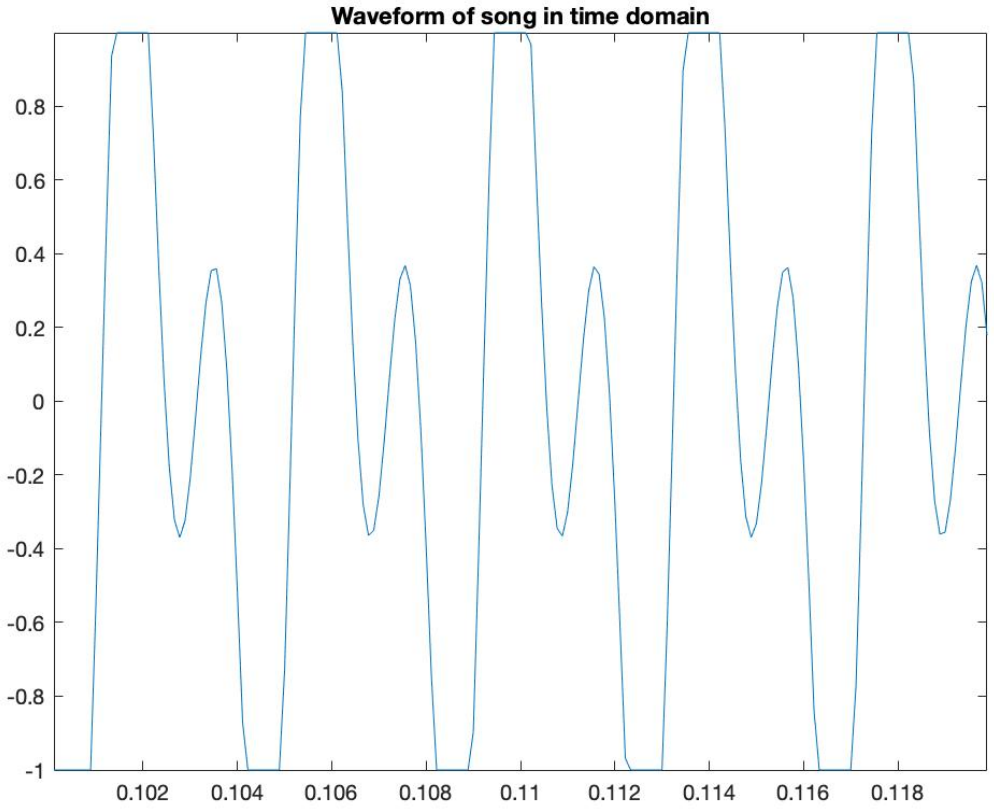
- Plot the time and frequency domain components of the music you created;
- Explain the output of your graphs according to your design.
- Plot the spectrogram of the music you created.

- Add white Gaussian noise to your music and analyze and plot the output music in both time and frequency domains. Explain the signal to noise ratio of your output music
- Design a filter to remove noise
- Analyze and plot the music in both time and frequency domain.

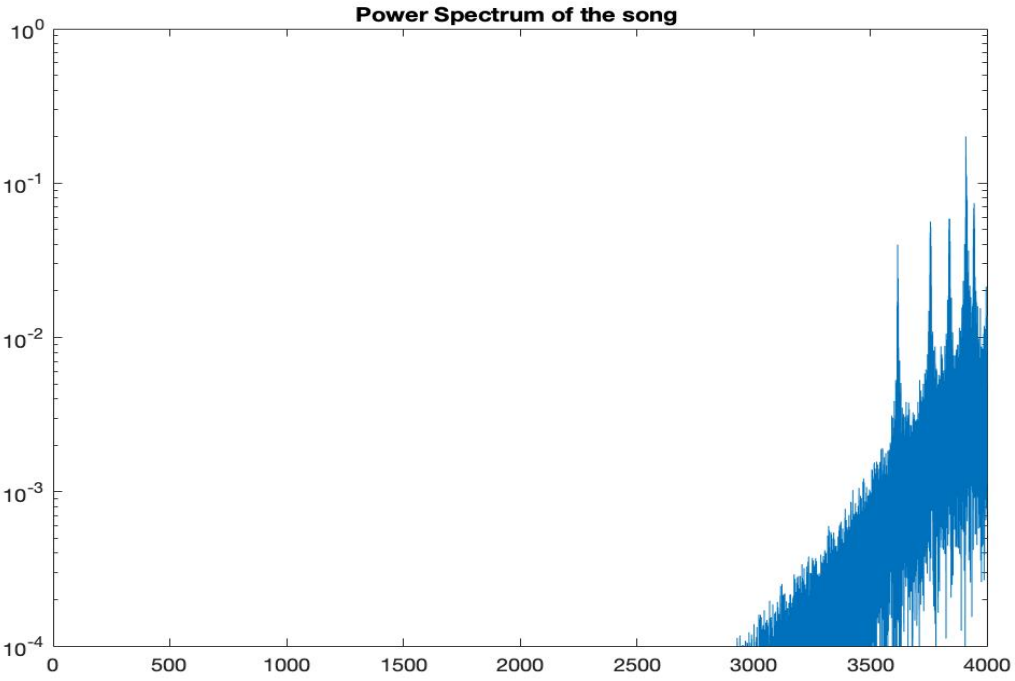
2 Results

As discussed earlier, the code generates a song using the MATLAB command audiowrite. This command creates a (.wav) wave sound file.

2.1 Generated output



Using the powerful range of tools offered by Matlab, I was able to plot the signal generated by the song within the real-time domain using the command audioread as seen in the graph above. I also plotted the Spectogram of the signal using the command semilogy resulting in the following.



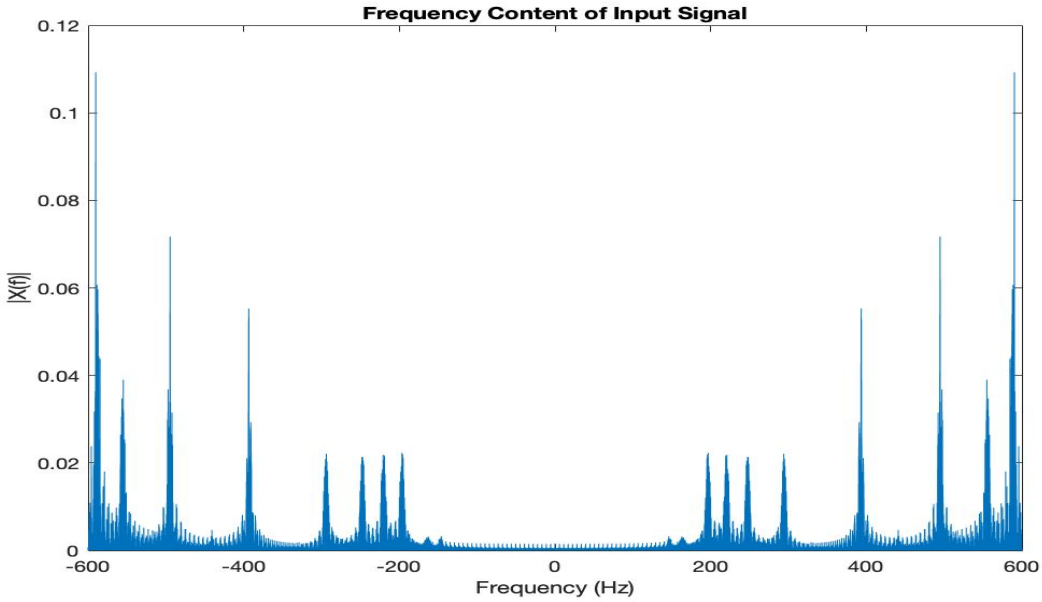
In the following section we compute for the Fast Fourier Transform signal and plot the time and frequency components.

2.2 Fast Fourier Transform

The relation between the sound data y and the frequency dependent c computed by Matlab fft command is:

$$y_j = 1/N \sum_{k=0}^{N} c_k e^{\iota 2\pi (j-1)(k-1)/N} \quad \text{for } k \in \{0, 1, \cdots, N\}$$

We calculate the Fast Fourier Transform of our song, then plot the frequency content of the input signal which then will be utilized to add the Gaussian noise. the figure below the frequency content of that signal.

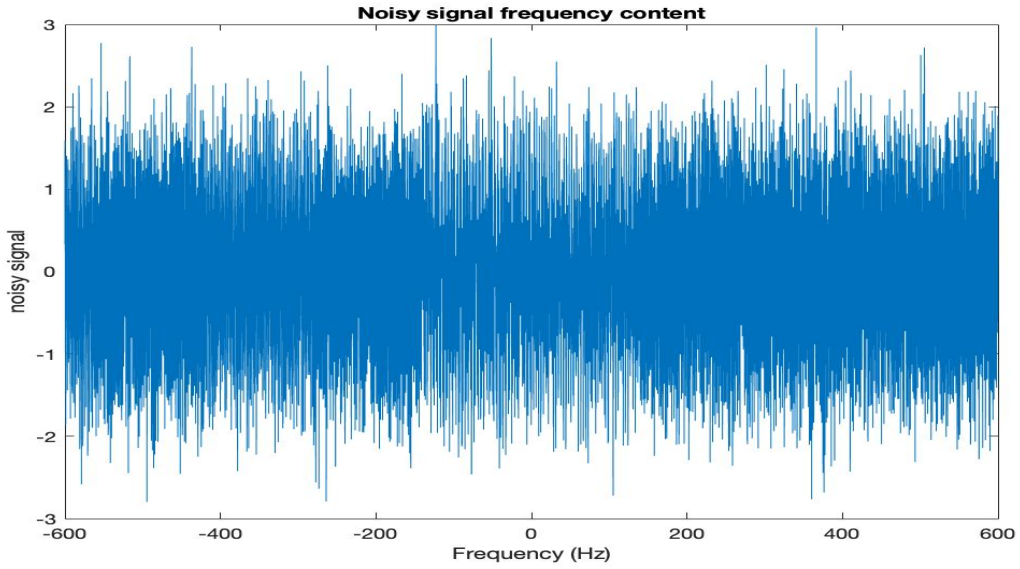


The computed value of c as referred to in the Appendix converts the sound data vector into another vector p representing the amplitude at each harmonic using the equation below; thus, the peaks shown in the above graph.

$$p_k = \sqrt{a_k^2 + b_k^2} = 2c_k$$

2.3 Gaussian Noise

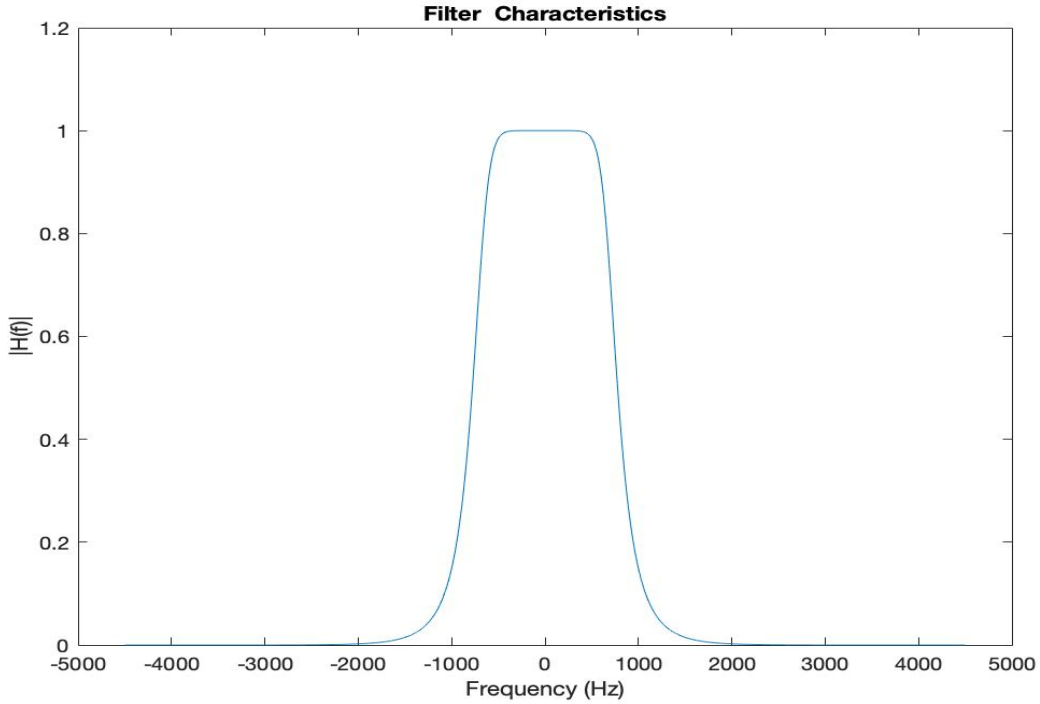
In order to add the Gaussian noise to the signal, the user needs to download the Simulink communications toolbox as an extension to Matlab. The packages include a command called awgn(in,snr) which take two arguments: in refers to the input signal and snr is the signal to noise ratio. Using the command audiowrite, a file gets generated from the resulting signal. By altering the parameters of the arguments, I was able to get quite a balanced output where both music and the noisy signal could be heard. The command audiowrite generates an audio file. The figure below shows the noisy signal generate with an snr value of 5.



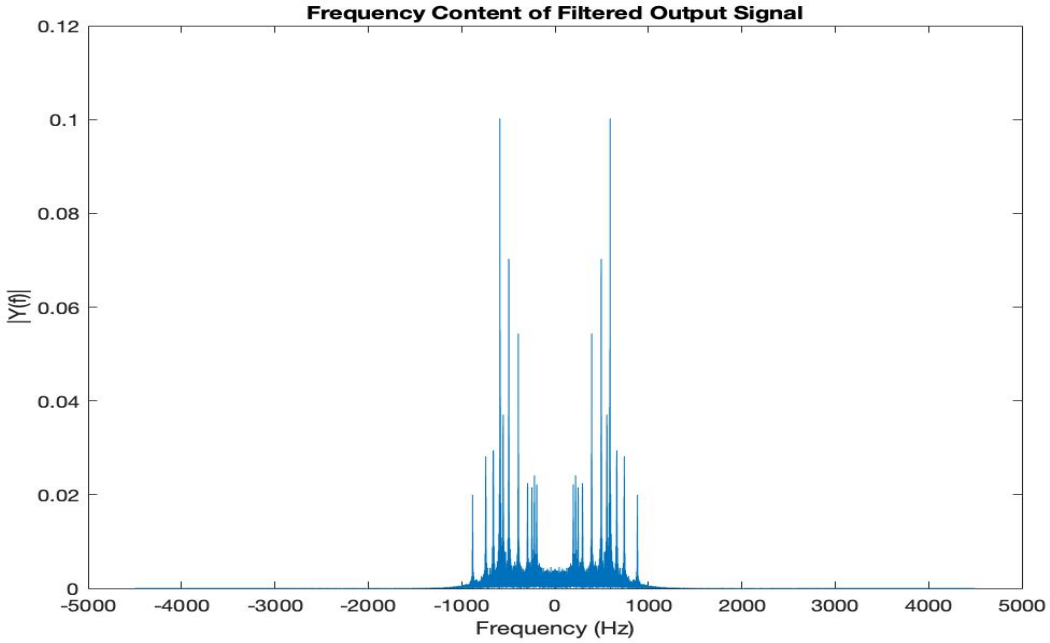
The x and y axis's represent the Frequency in (Hz) and the noisy signal, respectively.

2.4 Filtration process

For the purpose of filtering this noisy signal, I first started by designing a low pass filter. The resulting band-pass design is of order 5n. the plot below shows the characteristics of the filter used.



Then, by using the command y = filter(b, a, noise) and audio write, we generate a filtered audio signal that passed the low pass filter designed.



3 Conclusion and comments

This project has been instrumental in reaching some of my own goals as a student. With a healthy bit of nostalgia, I was able to reproduce the theme song of the famous game Mortal Kombat. I have a slight musical background but it is not a necessity to implement songs in Matlab. Moreover, I used concepts from Signals and Systems class like Fourier transform and the Mathematical approach to analyze the signal, shifting when necessary, and finally scaling the signal for convenience. The main challenges of this Project are digitization, using knowledge of simple signal processing, practice with tuning the frequency for sine wave functions to obtain a clear output, and working with large arrays to generate the desired result.

Appendix

```
close all
clear
clc

t = [0:.000125:.20];

Al = sin(2*pi*220*t); % The 'l' attached to A is to show it's a note below middle C.
A = sin(2*pi*440*t);
C = sin(2*pi*523.25*t);
D = sin(2*pi*587.33*t);
E = sin(2*pi*659.26*t);
midc = sin(2*pi*261.6*t);
G = sin(2*pi*783.99*t);
Gl = sin(2*pi*196*t); % The 'l' attached to 'G' is to show it is below middle C
B = sin(2*pi*493.88*t);
F = sin(2*pi*349.23*t);
Fl = sin(2*pi*174.61*t); % The 'l' attached to 'F' is to show it is below middle C
Rs = sin(2*pi*000*t); % I use this for a rest note
Dl = sin(2*pi*293.66*t); % The 'l' attached to the 'D' is to show it is below middle c
Gs = sin(2*pi*392*t); %This G note was not below middle c but not a higher not either.
Bl = sin(2*pi*246.94*t); % The 'l' attached to the 'B' is to show it is below middle c
El = sin(2*pi*164.81*t); % Same condition as above.
As = sin(2*pi*466.16*t); % The 's' stands for a # note

% These are notes that needs to be played at the same time.
% Using 'Var' = ('x' + 'y') I was able to do this.
AlA = (Al + A);
DAl = (D + Al);
Cmidc = (C + midc);
GGl = (G + Gl);
CGl = (C + Gl);
FFl = (F + Fl);
BFl = (B + Fl);
% This is where I create my 'measures' and lines.
line1 = [AlA,A,C,A,D,A,E,DAl,Cmidc,C,E,C,G,C,E,Cmidc,GGl,G,B,G,C,G,D,CGl];
line2 = [FFl,F,A,F,B,F,C,BFl];
```

```
line3 = [FF1,F,A,F,B,F,C,BF1];

% This is where I create my actual song. I organize my lines in the correct
% order according to the sheet music I used for the actual song.
song  =  [line1,line2,line1,line3];

audiowrite('MortalKombat.wav',song, 9000);
[x, fs] = audioread('MortalKombat.wav');
noise= awgn(x,5);
audiowrite('MortalKombatWithNoise.wav',noise, 9000);
N = length(x);
X = fft(x);
X = X/N;
X1 = fftshift(abs(X));
f = fs*linspace(0,1, N) - fs/2;

figure
% subplot(5, 1, 1)
plot(f,X1);
ylabel('|X(f)|'); xlabel('Frequency (Hz)')
title('Frequency Content of Input Signal')
xlim([-600 600])

figure()
% subplot(5, 1, 1)
plot(f, noise);
xlim([-600 600])
ylabel('noisy signal'); xlabel('Frequency (Hz)')
title('Noisy signal frequency content')

% build filter
% The code below shows an example on how to build and use a digital filter.
cutoff = 700; % cutoff frequency (Hz) for the filter
% build a 5th-order butterworth low pass digital filter
% to keep the music but filter out the noise
% b: numerator coeff of system transfer function, a: denominator coeff
% system transfer function is discussed in section 9.7 of the textbook.
% In the command below, 5 is the order of the filter, the second parameter
% specify the cutoff frequency (with value between 0 and 1), which is
% normalized to half the sample rate, 'low' denotes the type of the filter,
%  low pass filter
[b, a] = butter(5, 2*cutoff/(fs), 'low');
H = freqz(b, a, f, fs); % get frequency response of filter
figure()
plot(f, abs(H)) % plot frequency response (magnitude) of filter
ylabel('|H(f)|'); xlabel('Frequency (Hz)')
```

```
title('Filter  Characteristics')

% Signal processing: pass the input through the lowpass filter
y = filter(b, a, noise); % using difference equation
audiowrite('MortalKombatFiltered.wav',y,9000);

% plot frequency content of filter output
N = length(y);
Y = fft(y(1:N))/N; % generate FFT and divide by num of points to normalize FFT;
f = fs*linspace(0,1, N) - fs/2; % generate frequency vector
Y1 = fftshift(abs(Y)); % magnitude of the double-sided spectrum
figure
plot(f,Y1);
ylabel('|Y(f)|'); xlabel('Frequency (Hz)')
title('Frequency Content of Filtered Output Signal')

tt = (1:length(x))/fs;          % time
dur = find(tt>0.1 & tt<0.12);   % set time duration for waveform plot
figure
plot(tt(dur),x(dur))
axis tight
title('Waveform of song in time domain' )

p = 2*abs( Y); % compute power at each frequency
figure
semilogy(f,p)
axis([0 4000 10^-4 1])
title('Power Spectrum of the song ')  % generate spectogram
```