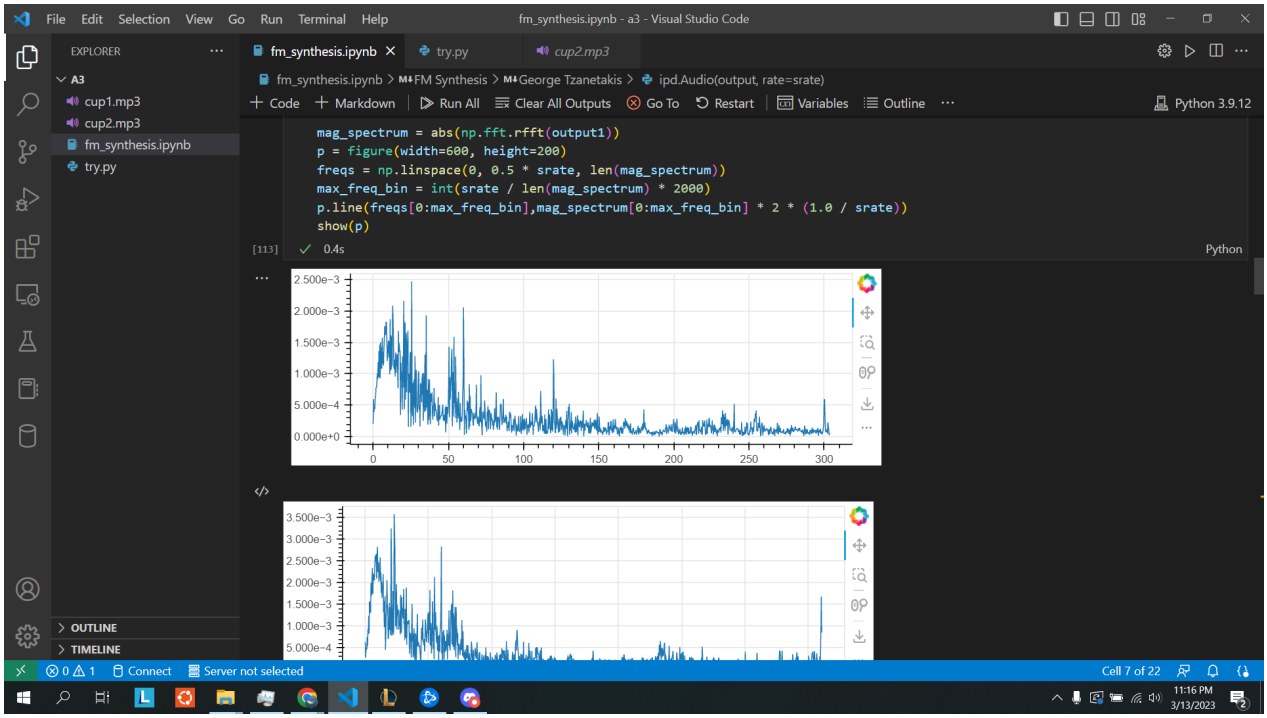
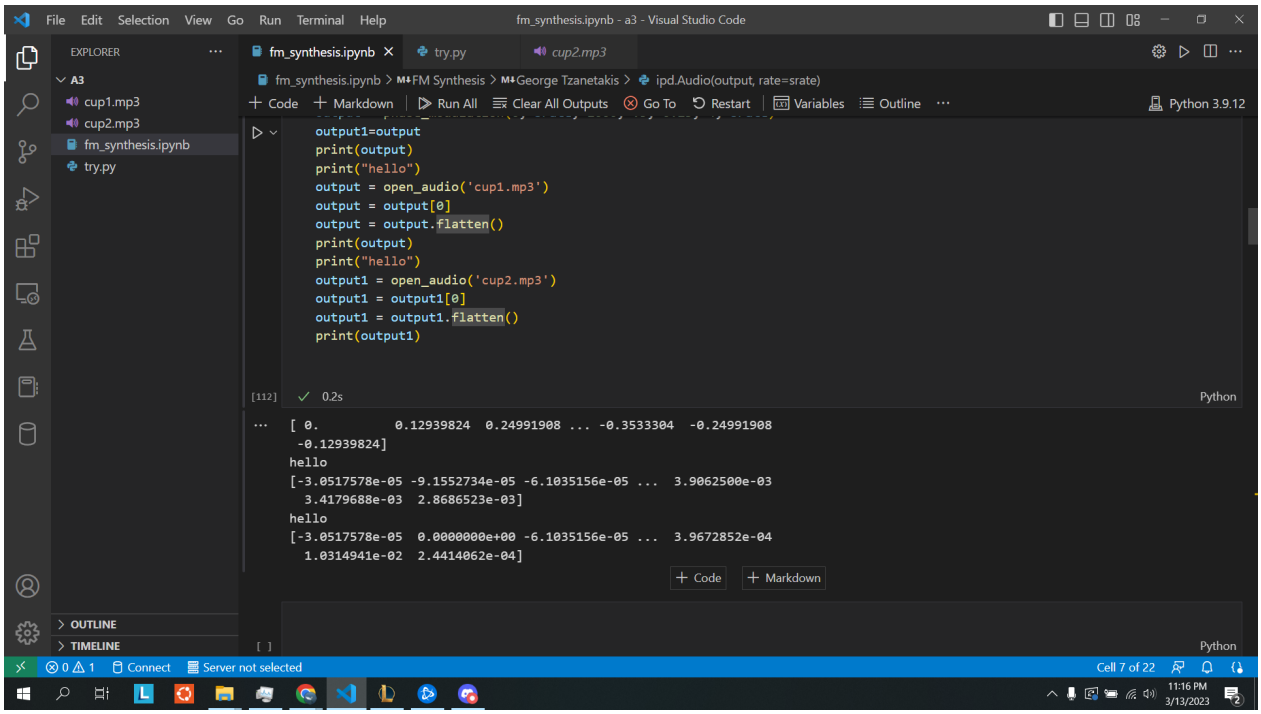
csc484D a3

For question 1 I watched a video called “The Roland D-50 - A Late 80's Love Story”. The 15-30 seconds I chose was the time period of 3:40-4:10 in the video. This video showed me the importance of the Roland D-50 through it’s historical presence through the groundbreaking and ahead of it’s time technology that was used in this synthesizer. Roland's linear arithmetic synthesis formula was combining the digital and analog synthesizer technologies to eliminate the cons of each. The digital sounded more realistic but the analog was easier to program but with this D-50 you could have both. The D-50 also was innovative in that it would allow it’s user to play up to 4 sounds at the same time. Unlike previous machines that would only have square and ramp waveshapes, samples of real sounds could be played with the D-50. Other synthesizers had something similar in stored waveforms, but the D-50 played actual digital recordings. The D-50 revolutionized the industry and without it we might not be at the advancement we are at today and this class might not even exist.

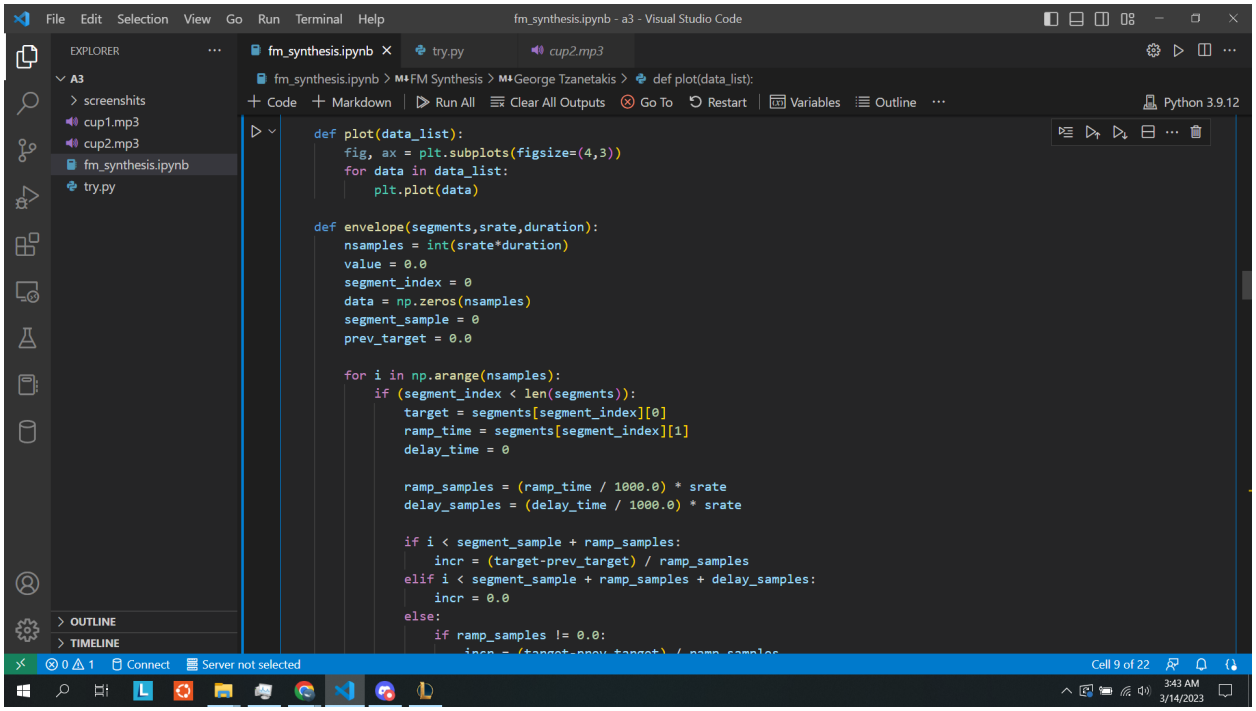
For question 2 I did not see any interviews on the resources page so I went to youtube and watched an interview in a youtube video called “1986 Computer Music (Synthesizers, Synclavier) News Report”. The 30-60 seconds I chose was the time period of 5:50-6:50 in the video. In terms of the history of music, Frequency Modulation is a big step that gets the human race to where it is musically and technologically today. This new method of synthesis was altering the synthesizer’s place in music. Synthesizers had been stuck in a certain role of making specific sounds that were unique to synthesizers but it had setbacks due to limited versatility in terms of sounds and genres. But with these new synthesizers with frequency modulation, that versatility problem was not an issue anymore. The sounds that were generated with the synthesizers that had frequency modulation could replicate other instruments but could not replicate them perfectly. Even though it was not perfect, this technology outlined the future of where synthesizers would go and had to go to be able to compete in the market.

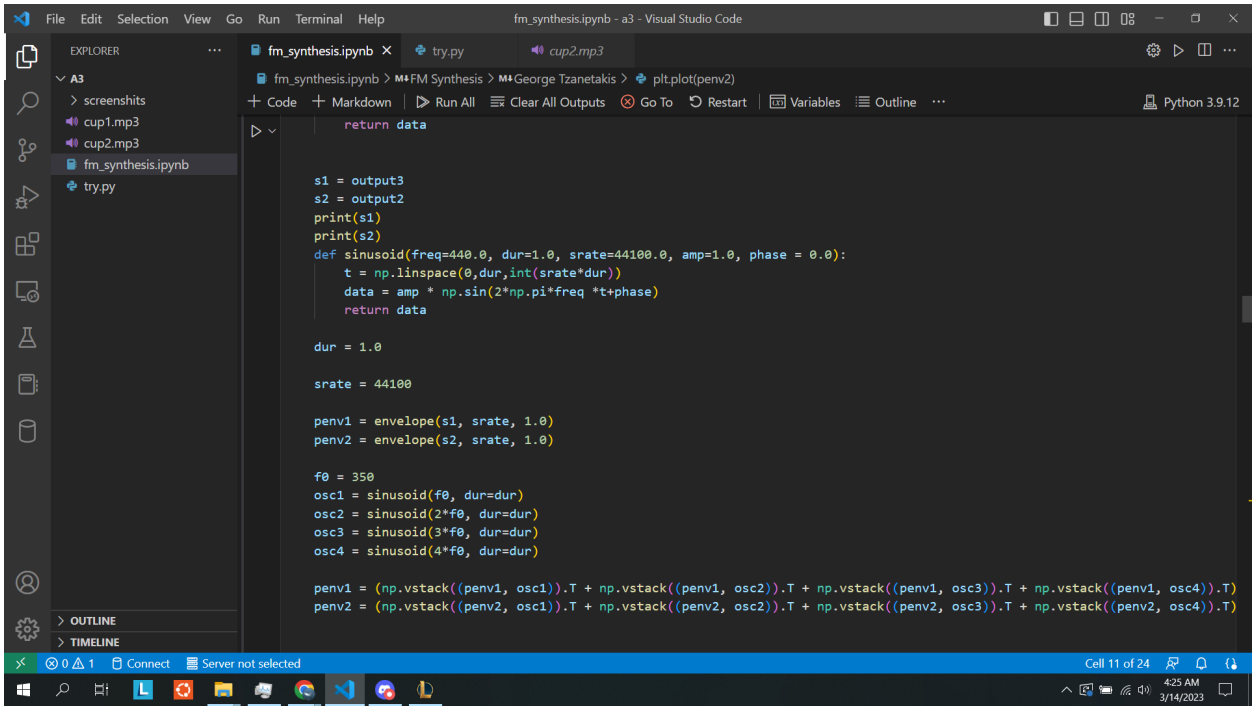


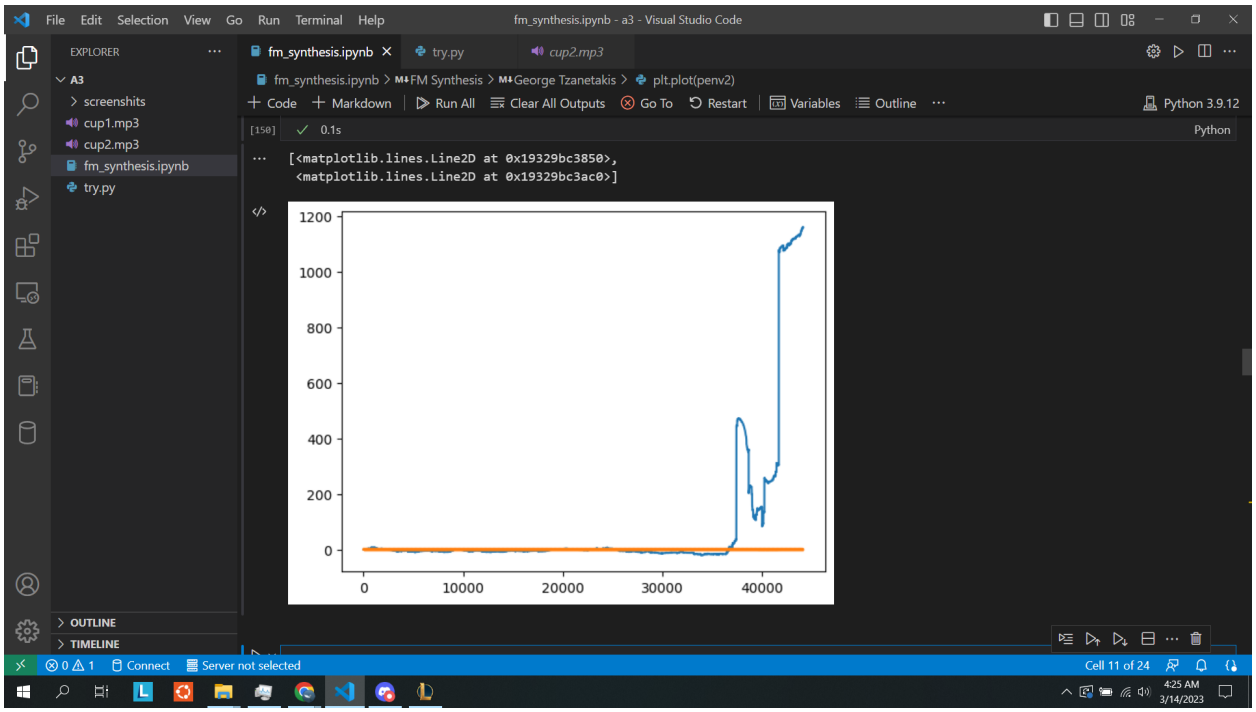


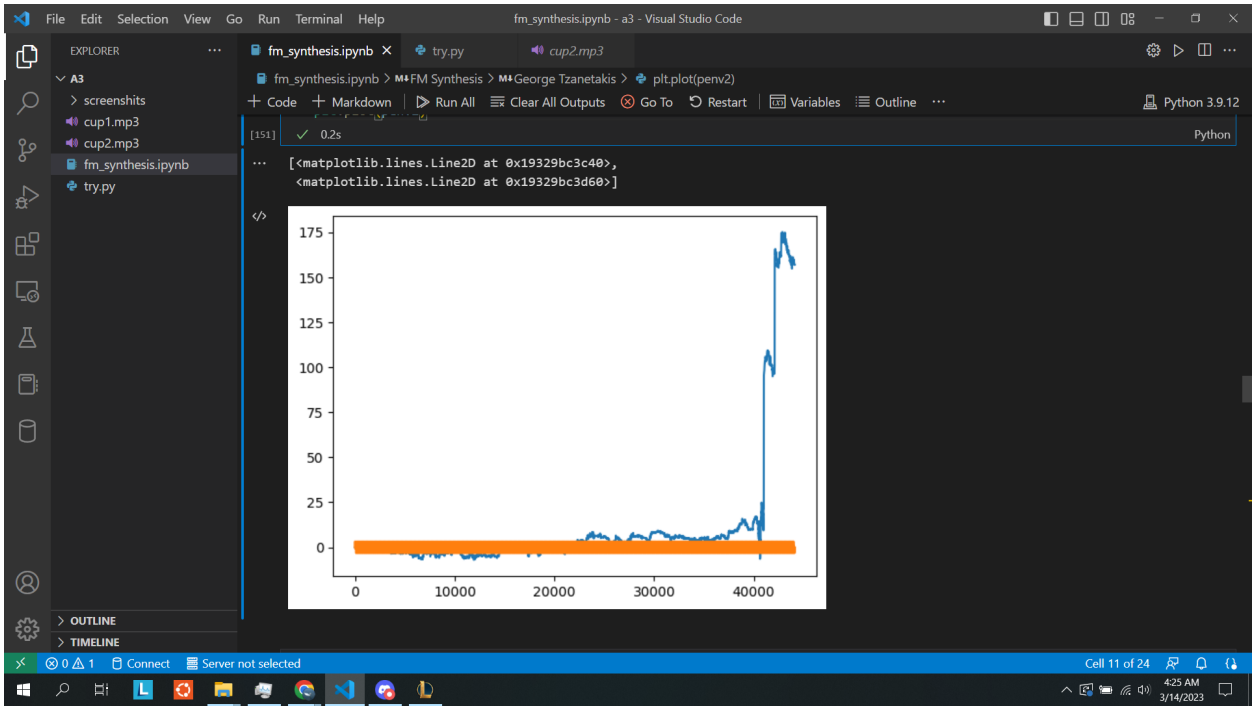
Question 3 asks me to plot the corresponding magnitude spectra using two coffee mugs and a pen, I didn't read the question correctly and hit the mugs with a metal spoon so the plots may look a little different than they are supposed to. I used most of the code for this question in plotting magnitude spectra in Python from the FM synthesis notebook on the github. The first thing I did was convert the mp3 files of me hitting the mugs into audio arrays. The arrays come in as 2d arrays but the code for the magnitude spectra plot required the array to be in a 1d array, for this I flattened the arrays so that they would meet the requirements. I had to code the same

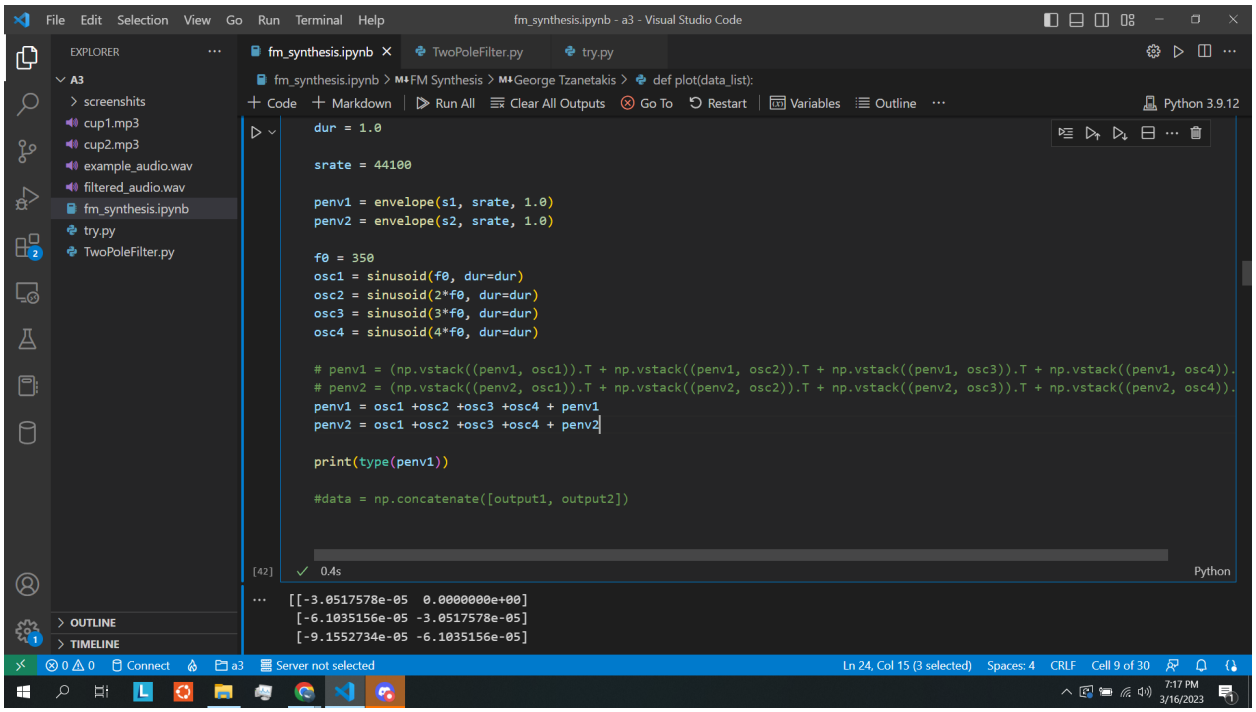
thing again for the second audio file. And then I got the resulting 2 plots. I listened to the flattened audio files and they just sounded like a slowed down version of themselves.

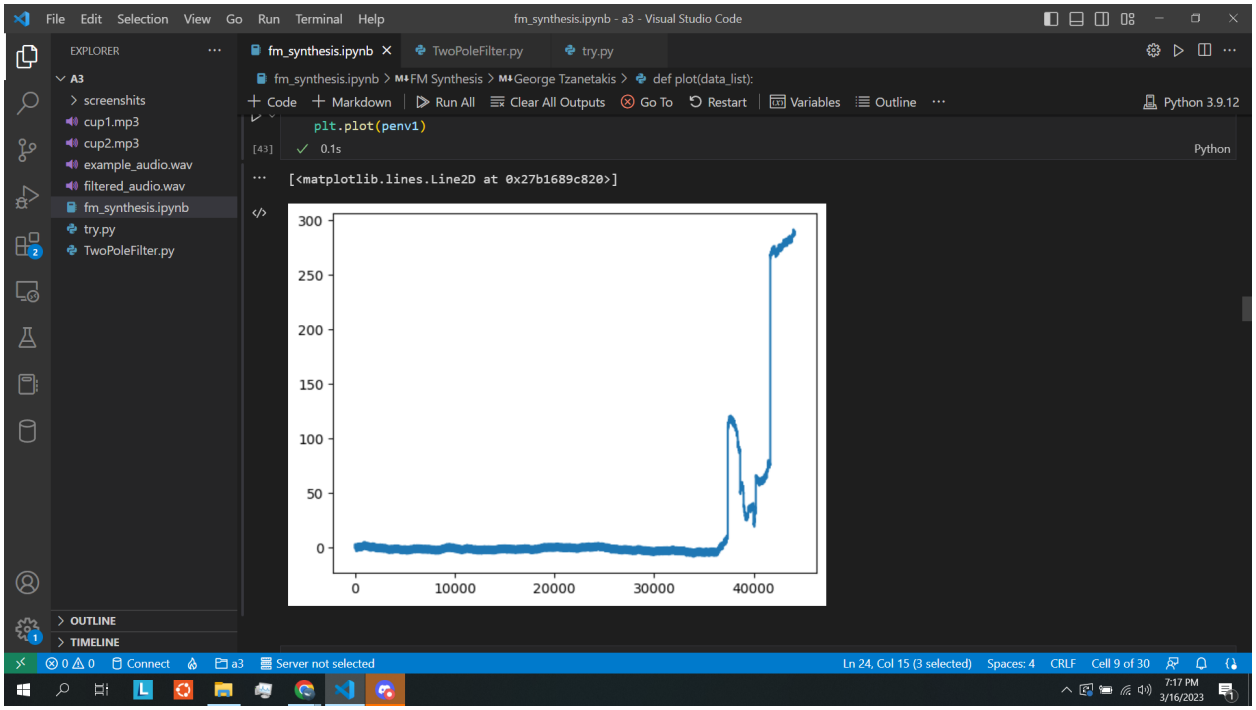


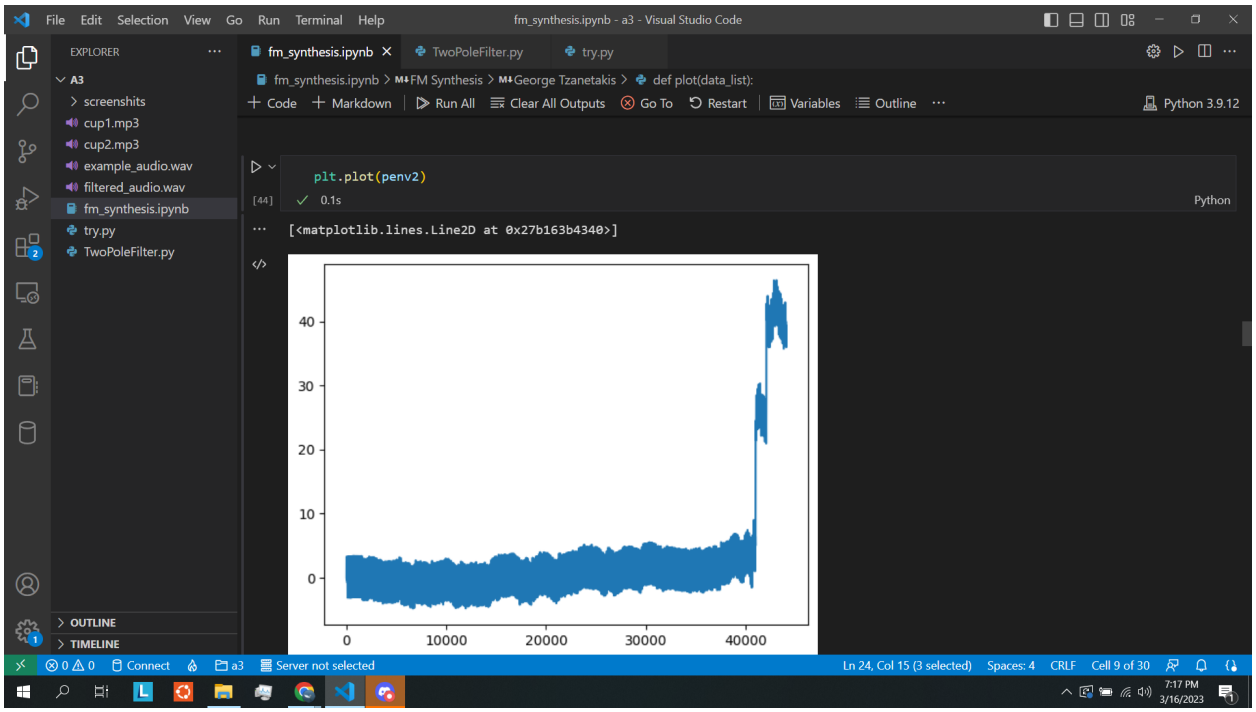

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
fm_synthesis.ipynb
try.py
fm_synthesis.ipynb
try.py
cup2.mp3
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > plt.plot(penv2)
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
return data
s1 = output3
s2 = output2
print(s1)
print(s2)
def sinusoid(freq=440.0, dur=1.0, srate=44100.0, amp=1.0, phase = 0.0):
t = np.linspace(0,dur,int(srate*dur))
data = amp * np.sin(2*np.pi*freq *t+phase)
return data
dur = 1.0
srate = 44100
penv1 = envelope(s1, srate, 1.0)
penv2 = envelope(s2, srate, 1.0)
f0 = 350
osc1 = sinusoid(f0, dur=dur)
osc2 = sinusoid(2*f0, dur=dur)
osc3 = sinusoid(3*f0, dur=dur)
osc4 = sinusoid(4*f0, dur=dur)
penv1 = (np.vstack((penv1, osc1)).T + np.vstack((penv1, osc2)).T + np.vstack((penv1, osc3)).T + np.vstack((penv1, osc4)).T)
penv2 = (np.vstack((penv2, osc1)).T + np.vstack((penv2, osc2)).T + np.vstack((penv2, osc3)).T + np.vstack((penv2, osc4)).T)
OUTLINE
TIMELINE
0 1 Connect Server not selected
Cell 11 of 24
4:25 AM
3/14/2023


File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
fm_synthesis.ipynb
try.py
fm_synthesis.ipynb
try.py
cup2.mp3
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > plt.plot(penv2)
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
[150] 0.1s
Python
[<matplotlib.lines.Line2D at 0x19329bc3850>,
<matplotlib.lines.Line2D at 0x19329bc3ac0>]
1200
1000
800
600
400
200
0
0
10000
20000
30000
40000
OUTLINE
TIMELINE
0 1 Connect Server not selected
Cell 11 of 24
4:25 AM
3/14/2023


File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
fm_synthesis.ipynb
try.py
fm_synthesis.ipynb
try.py
cup2.mp3
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > plt.plot(penv2)
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
[151] 0.2s
Python
[<matplotlib.lines.Line2D at 0x19329bc3c40>,
<matplotlib.lines.Line2D at 0x19329bc3d60>]
175
150
125
100
75
50
25
0
0
10000
20000
30000
40000
OUTLINE
TIMELINE
0 1 Connect Server not selected
Cell 11 of 24
4:25 AM
3/14/2023

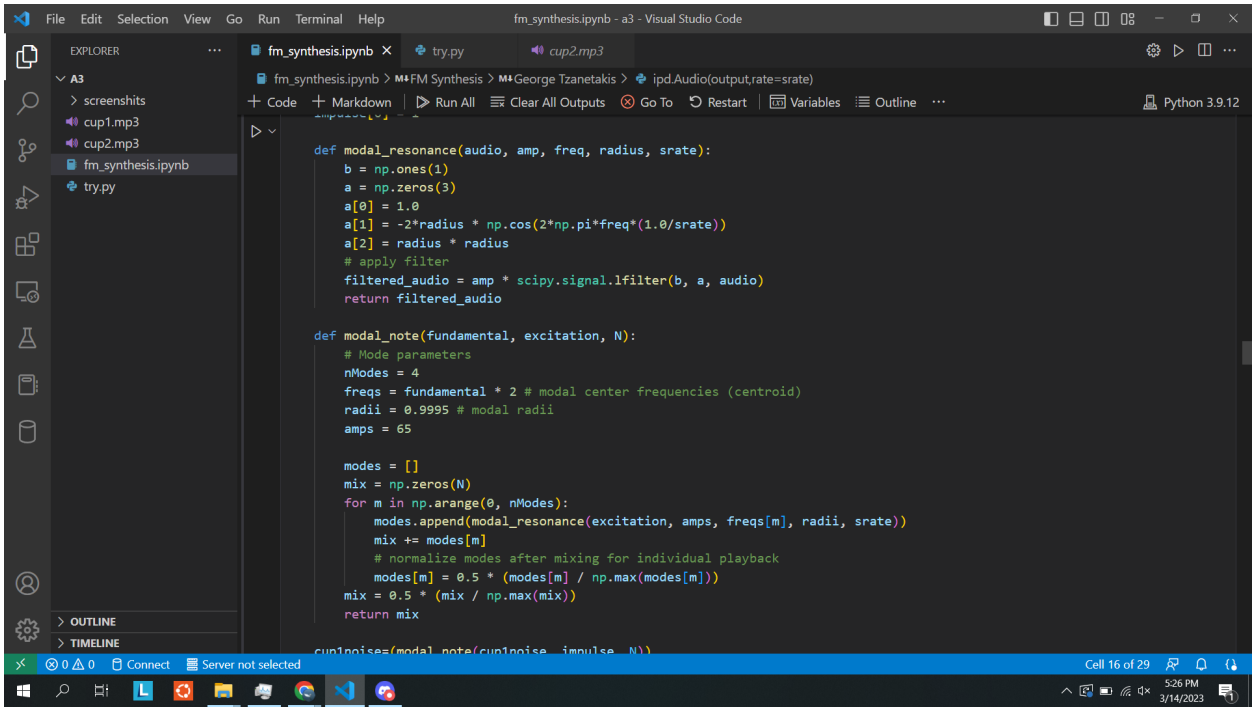

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > def plot(data_list):
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
dur = 1.0
srate = 44100
penv1 = envelope(s1, srate, 1.0)
penv2 = envelope(s2, srate, 1.0)
f0 = 350
osc1 = sinusoid(f0, dur=dur)
osc2 = sinusoid(2*f0, dur=dur)
osc3 = sinusoid(3*f0, dur=dur)
osc4 = sinusoid(4*f0, dur=dur)
# penv1 = (np.vstack((penv1, osc1)).T + np.vstack((penv1, osc2)).T + np.vstack((penv1, osc3)).T + np.vstack((penv1, osc4)).
# penv2 = (np.vstack((penv2, osc1)).T + np.vstack((penv2, osc2)).T + np.vstack((penv2, osc3)).T + np.vstack((penv2, osc4)).
penv1 = osc1 +osc2 +osc3 +osc4 + penv1
penv2 = osc1 +osc2 +osc3 +osc4 + penv2
print(type(penv1))
#data = np.concatenate([output1, output2])
[42] 0.4s
Python
[[-3.0517578e-05 0.0000000e+00]
[-6.1035156e-05 -3.0517578e-05]
[-9.1552734e-05 -6.1035156e-05]
OUTLINE
TIMELINE
0 0 Connect a3 Server not selected
Ln 24, Col 15 (3 selected) Spaces: 4 CRLF Cell 9 of 30
7:17 PM
3/16/2023

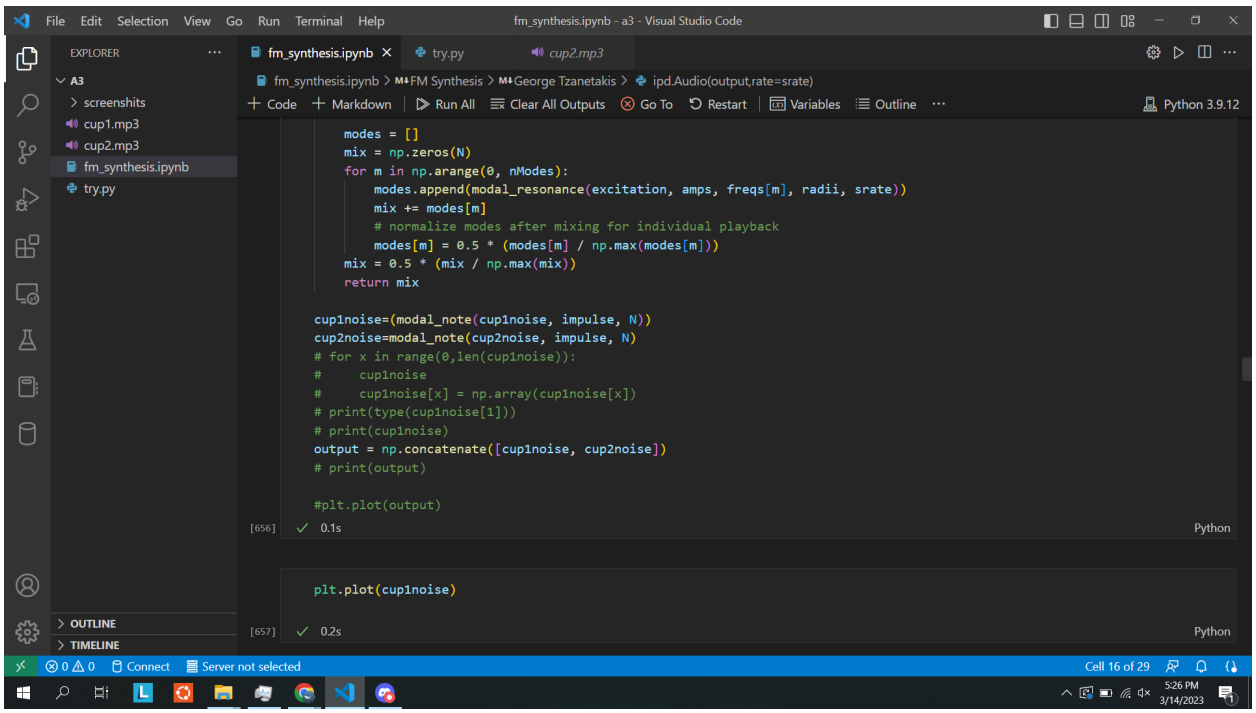


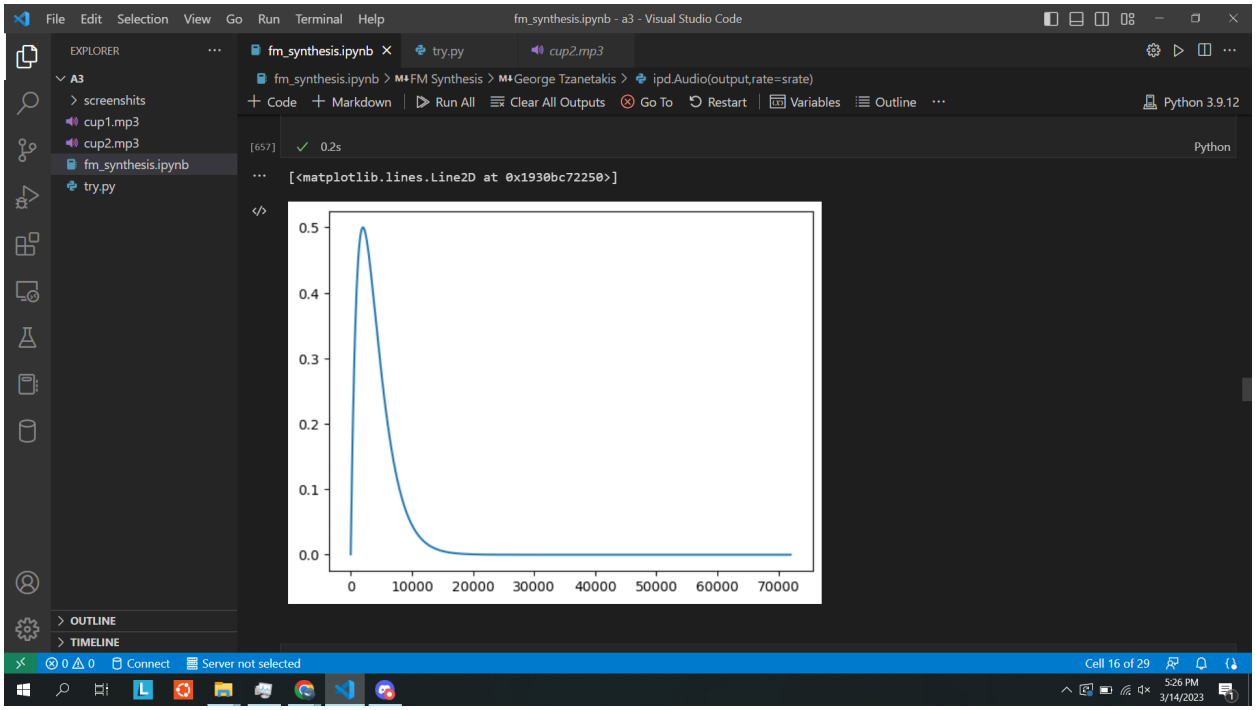


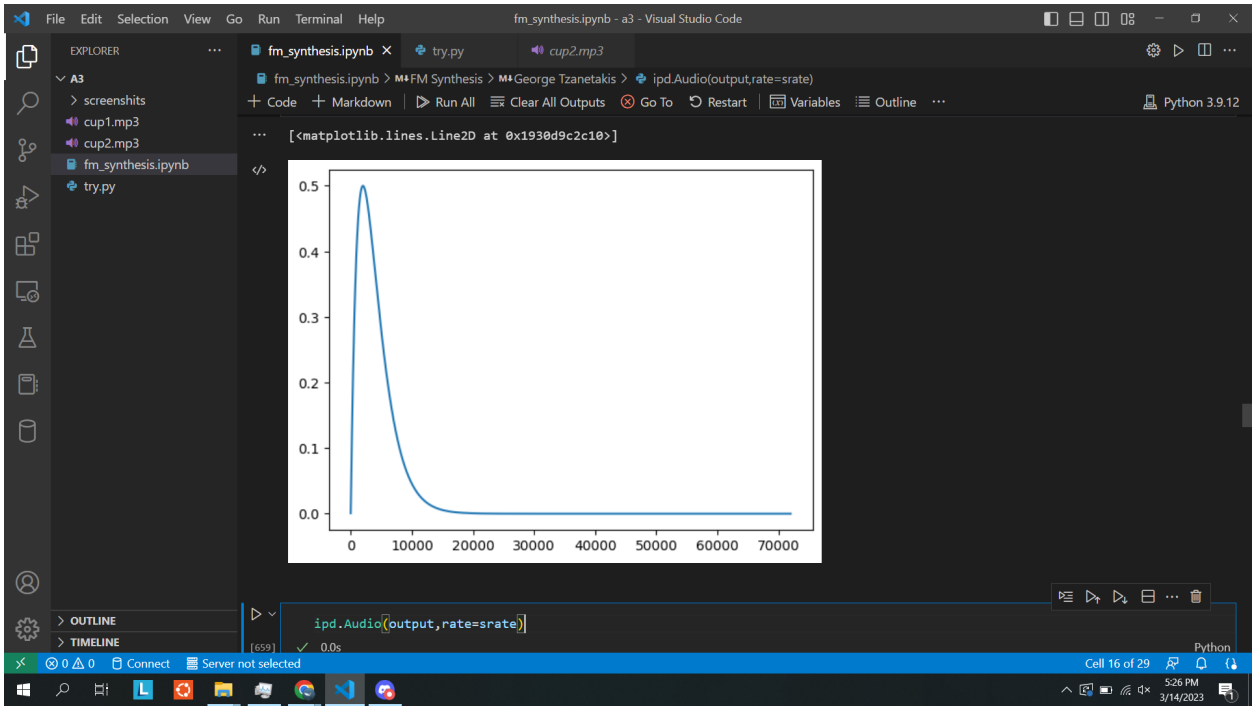
For Question 4 we had to make some additive synthesis models based on the coffee mug and spoon recordings. I did this in python, and had a little trouble making sure each model used 4 sinusoidal oscillators. The associated envelopes were a little easier as I looked through the notebook on the github. The issue I was having with the oscillators was that they would not go together due to size constraints and if I managed to make them go together the program would run for a long time. I ended up with graphs that look like this because I used vstack instead of dot product or multiply when putting the two arrays together. Overall it should have a similar

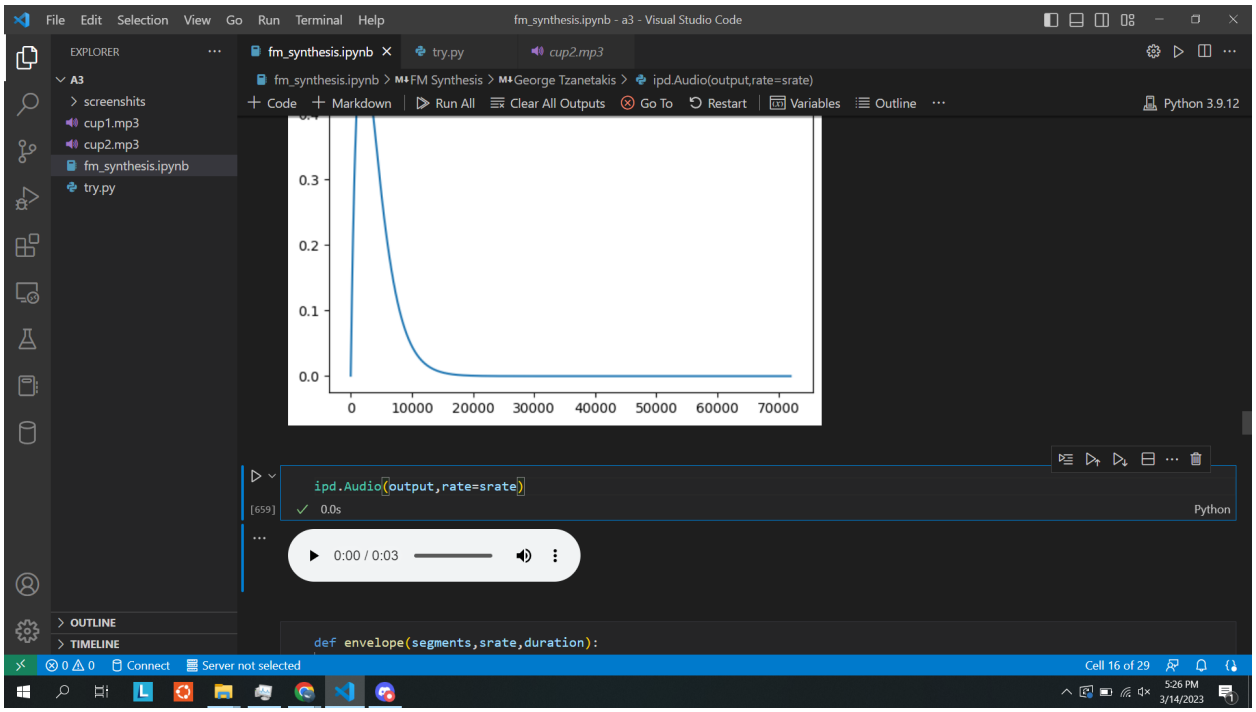
shape as the blue line though. When I listened to the resulting sound and I knew that one of the noises was a higher pitch than the other one but I could not tell which one was which when they were concatinated for the audio. Edit: based on what someone said in the discord I have slightly altered my code so instead of using vstack I am just adding the oscillators to the envelope directly. The screenshots for this change and the new graph plots are shown in the last 3 screenshots.






File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
fm_synthesis.ipynb
try.py
fm_synthesis.ipynb
try.py
cup2.mp3
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > ipd.Audio(output,rate=srate)
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
[657] 0.2s
Python
[<matplotlib.lines.Line2D at 0x1930bc72250>]
0.5
0.4
0.3
0.2
0.1
0.0
0
10000
20000
30000
40000
50000
60000
70000
OUTLINE
TIMELINE
0 0 Connect Server not selected
Cell 16 of 29
5:26 PM
3/14/2023

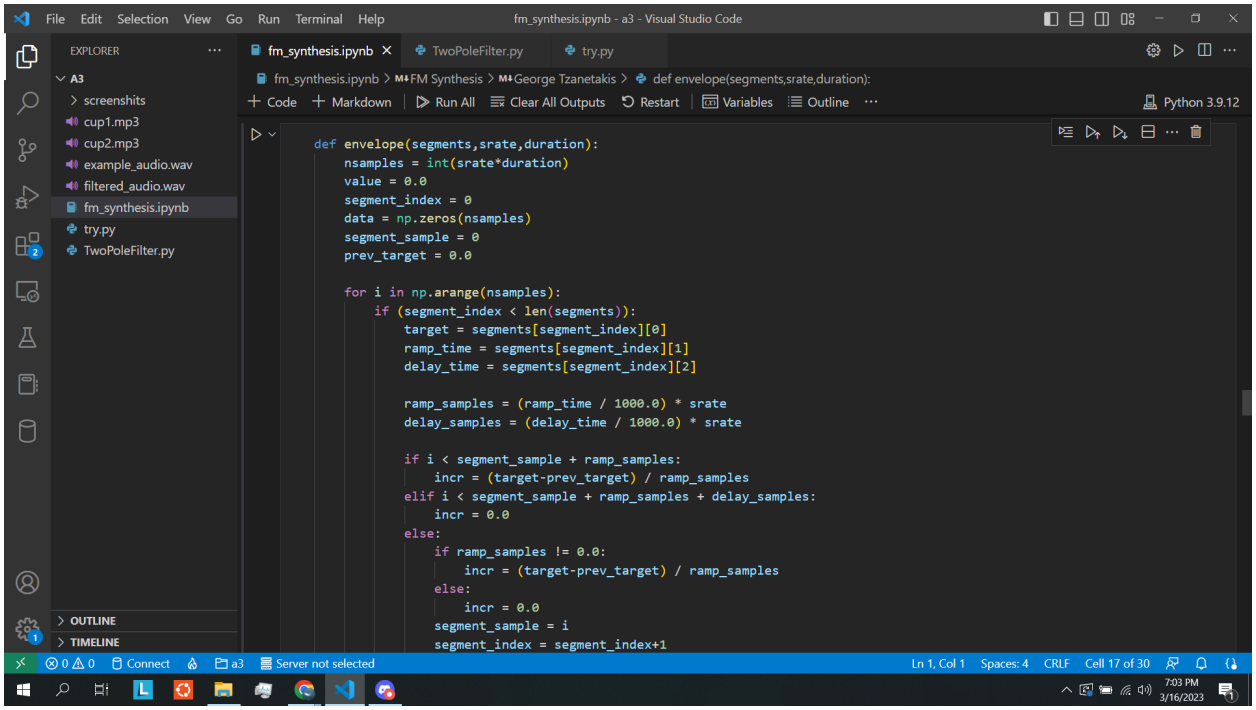


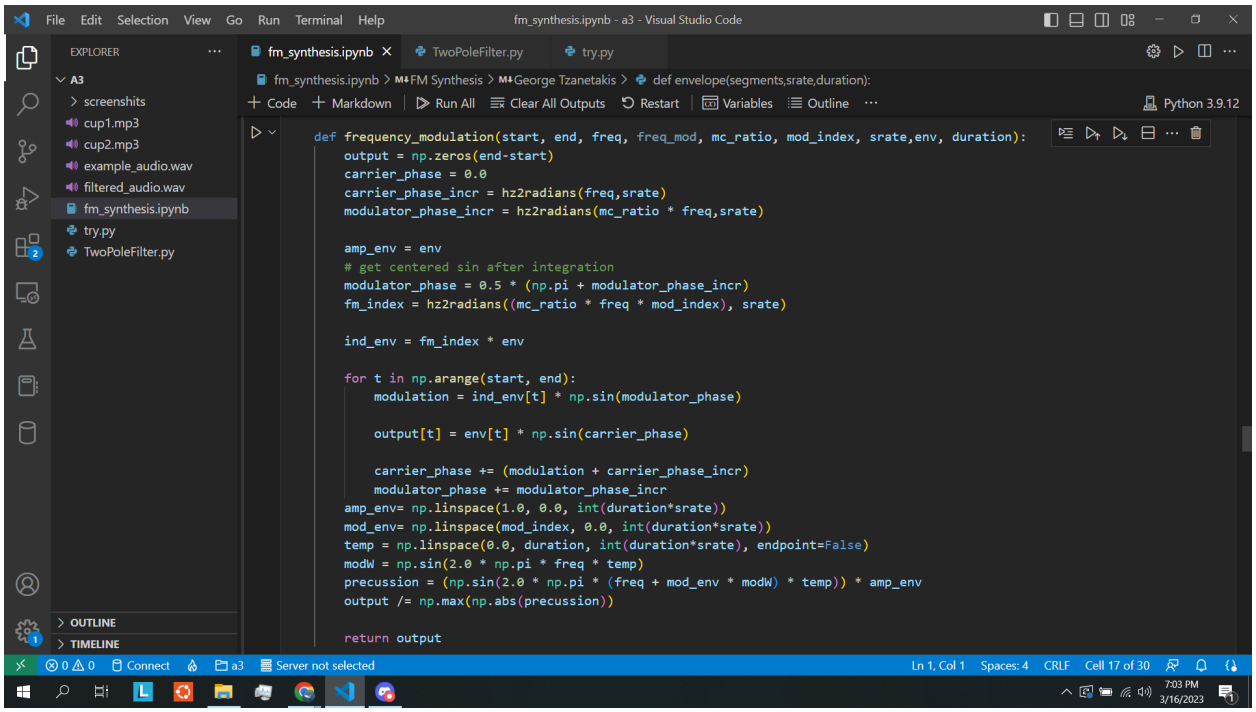


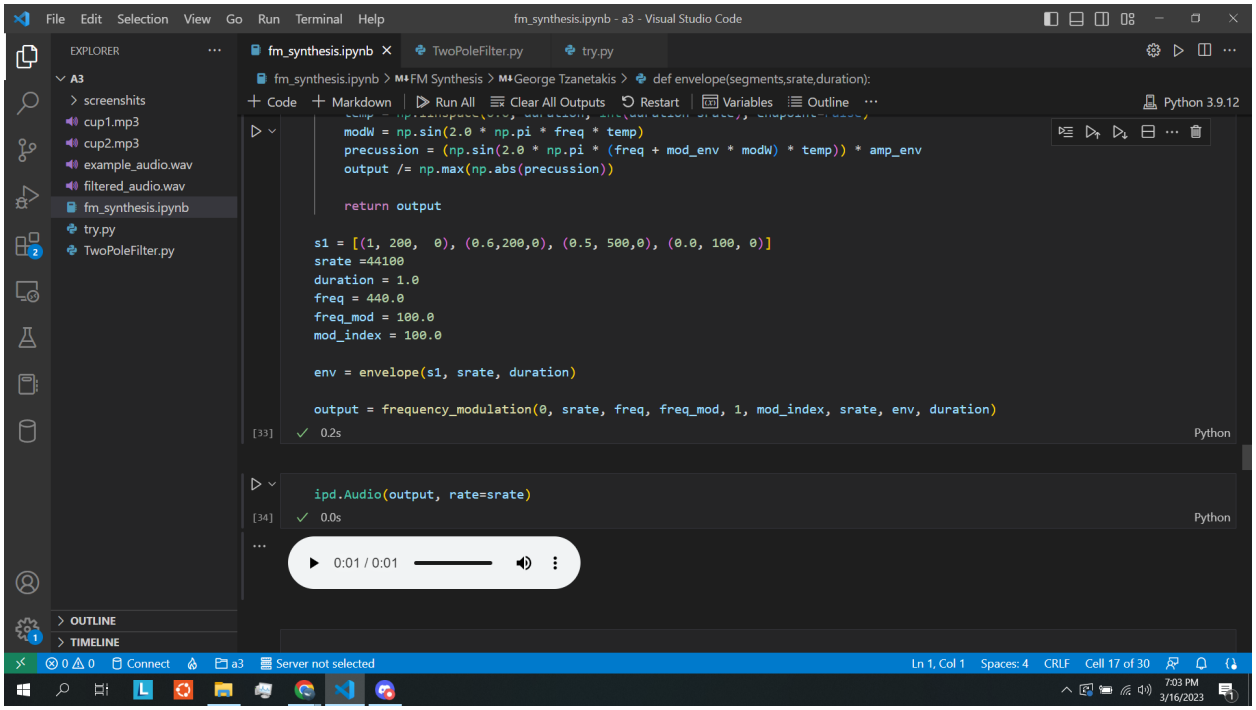
For this question I had to make two modal synthesis models based on the coffee mug recordings. Each model has 4 BiQuad filters but I had some problems with gauging the center frequencies and resonances. This is because with the center frequencies and resonances being an array, it takes my program a super long time to run, to make it more simple so that my program would run in a good time I altered the values so that i get the median of each of the arrays and use that value. When listen to the resulting sound I have a similar issue of I know

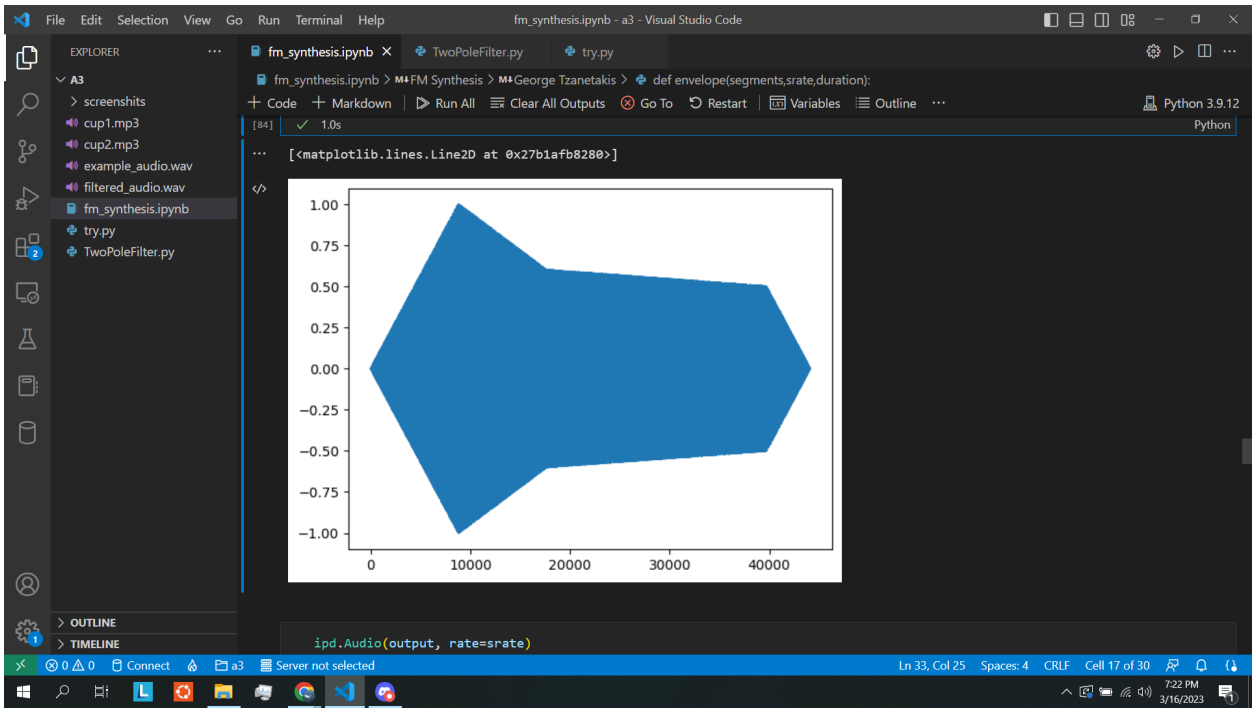
that one of the mugs made a higher pitched note than the other and I can kind of hear that but I don’t know which is which.

Question 6 asks me to compare magnitude spectra, additive synthesis approximation, and modal synthesis approximation. The magnitude spectrum of the original recording has a unique signature that shows the distribution of audio levels across the frequencies. When comparing the sounds of the original recording using additive synthesis approximation and modal synthesis approximation, the additive synthesis approximation will sound like a simple tone, whereas the modal synthesis approximation will have a more complex and resonant sound that captures the natural behavior of the mugs.The differences in the sounds and Python code of the magnitude spectra, additive synthesis approximation, and modal synthesis approximation would be the general process that involves analyzing the frequency content of the audio recordings.


File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > def envelope(segments,srate,duration):
Code Markdown Run All Clear All Outputs Restart Variables Outline
Python 3.9.12
def envelope(segments,srate,duration):
nsamples = int(srate*duration)
value = 0.0
segment_index = 0
data = np.zeros(nsamples)
segment_sample = 0
prev_target = 0.0
for i in np.arange(nsamples):
if (segment_index < len(segments)):
target = segments[segment_index][0]
ramp_time = segments[segment_index][1]
delay_time = segments[segment_index][2]
ramp_samples = (ramp_time / 1000.0) * srate
delay_samples = (delay_time / 1000.0) * srate
if i < segment_sample + ramp_samples:
incr = (target-prev_target) / ramp_samples
elif i < segment_sample + ramp_samples + delay_samples:
incr = 0.0
else:
if ramp_samples != 0.0:
incr = (target-prev_target) / ramp_samples
else:
incr = 0.0
segment_sample = i
segment_index = segment_index+1
OUTLINE
TIMELINE
0 0 Connect a3 Server not selected
Ln 1, Col 1 Spaces: 4 CRLF Cell 17 of 30
7:03 PM
3/16/2023

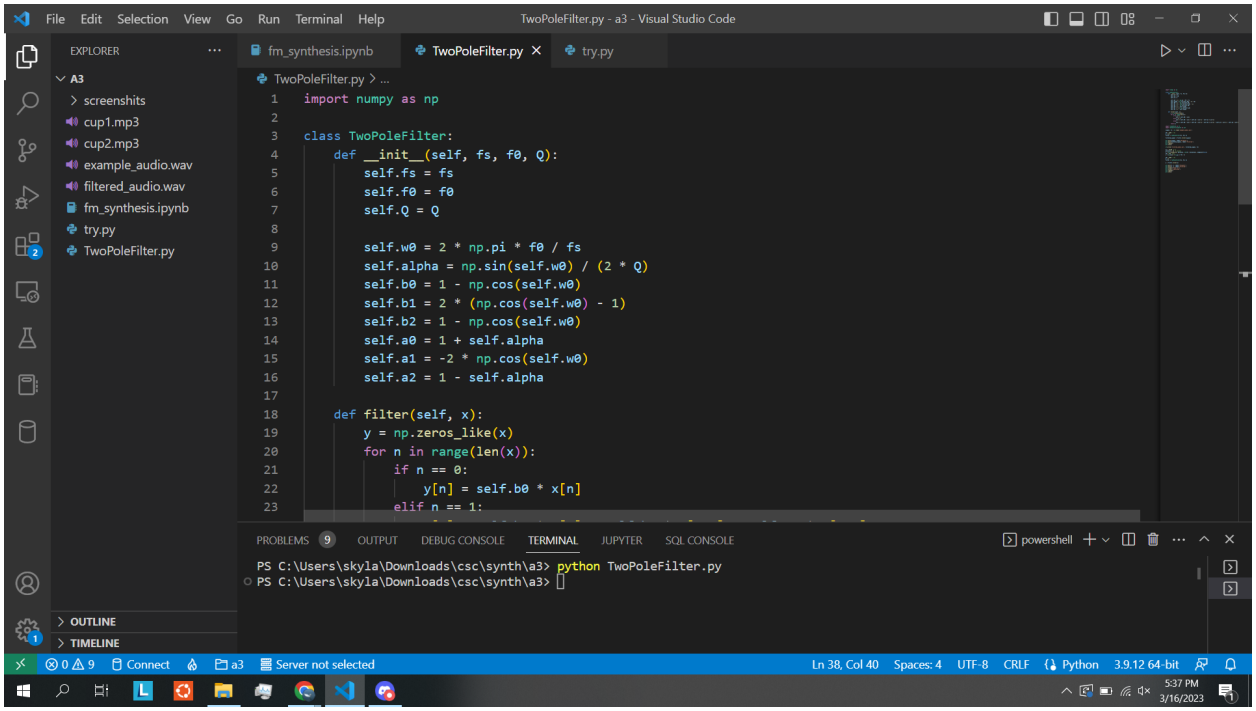

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > def envelope(segments,srate,duration):
Code Markdown Run All Clear All Outputs Restart Variables Outline
Python 3.9.12
def frequency_modulation(start, end, freq, freq_mod, mc_ratio, mod_index, srate,env, duration):
output = np.zeros(end-start)
carrier_phase = 0.0
carrier_phase_incr = hz2radians(freq,srate)
modulator_phase_incr = hz2radians(mc_ratio * freq,srate)
amp_env = env
# get centered sin after integration
modulator_phase = 0.5 * (np.pi + modulator_phase_incr)
fm_index = hz2radians((mc_ratio * freq * mod_index), srate)
ind_env = fm_index * env
for t in np.arange(start, end):
modulation = ind_env[t] * np.sin(modulator_phase)
output[t] = env[t] * np.sin(carrier_phase)
carrier_phase += (modulation + carrier_phase_incr)
modulator_phase += modulator_phase_incr
amp_env= np.linspace(1.0, 0.0, int(duration*srate))
mod_env= np.linspace(mod_index, 0.0, int(duration*srate))
temp = np.linspace(0.0, duration, int(duration*srate), endpoint=False)
modW = np.sin(2.0 * np.pi * freq * temp)
precussion = (np.sin(2.0 * np.pi * (freq + mod_env * modW) * temp)) * amp_env
output /= np.max(np.abs(precussion))
return output
OUTLINE
TIMELINE
0 0 Connect a3 Server not selected
Ln 1, Col 1 Spaces: 4 CRLF Cell 17 of 30
7:03 PM
3/16/2023

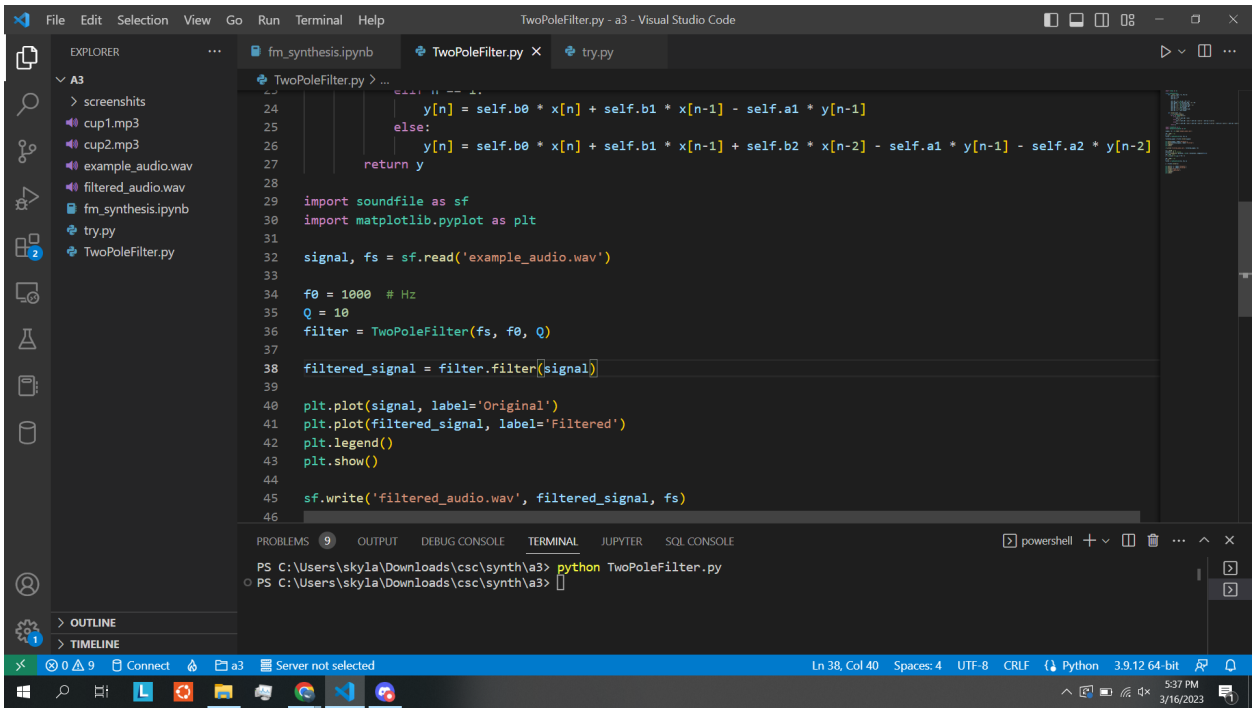


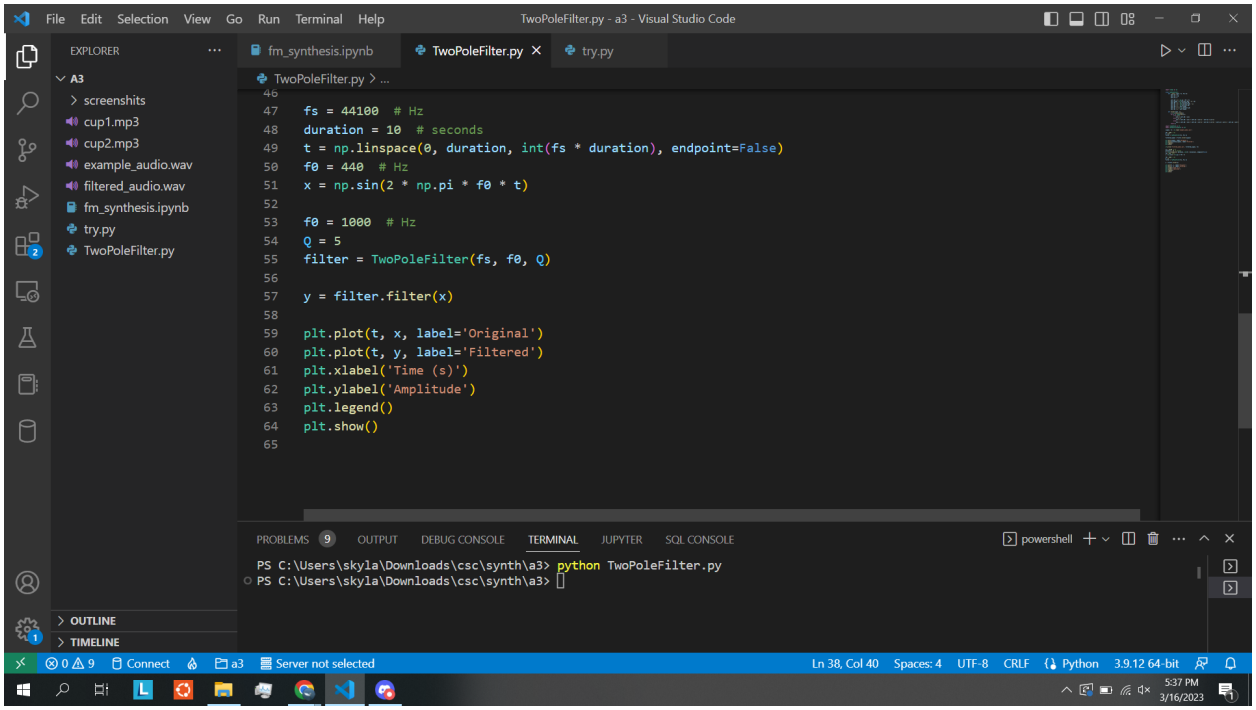


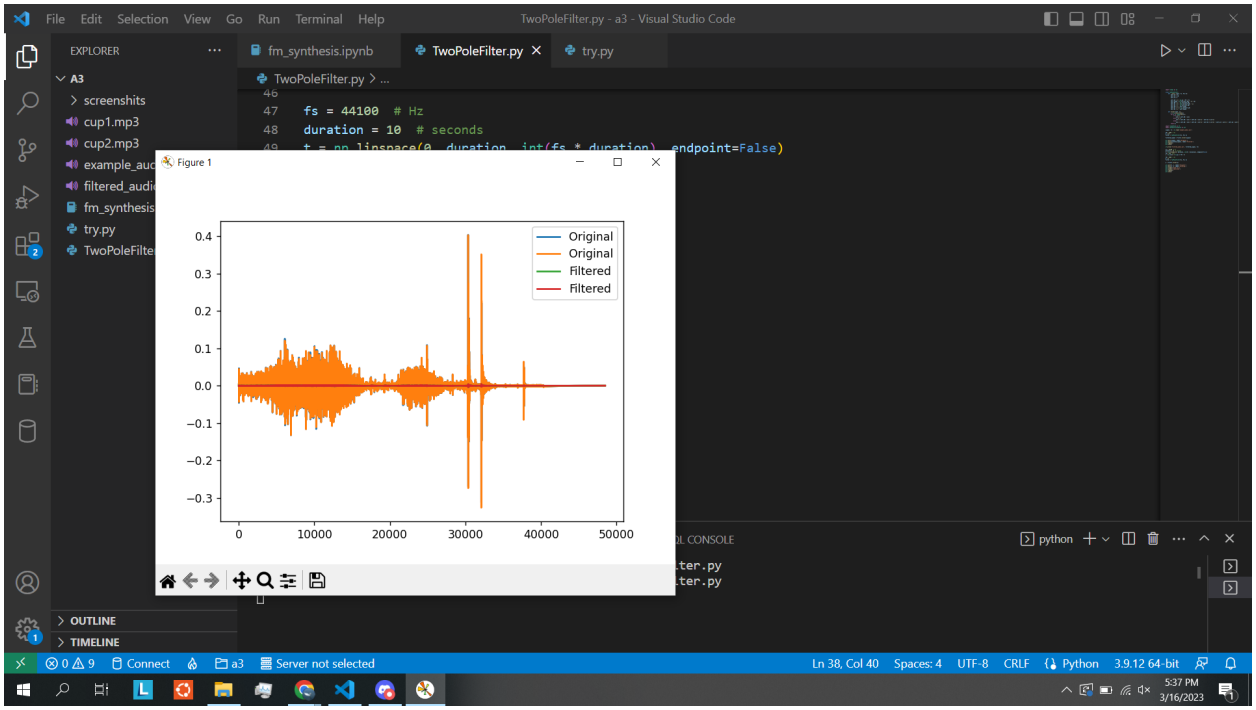
For question 7 I had to synthesize a percussive sound using FM synthesis. The attack is short and there is a high modulation index of 100. The high modulation index is probably what is causing the plot graph to look like it does in the last screenshot. About half of the code is similar to the fm synthesis notebook but the other half is what i have added. The envelope function is about the same with some minor tweaks. When going into the frequency_modulation function I added another variable so that I could track extra information and use the high modulation index. I could have shortened the attack more based on what is shown in the plot graph and it is

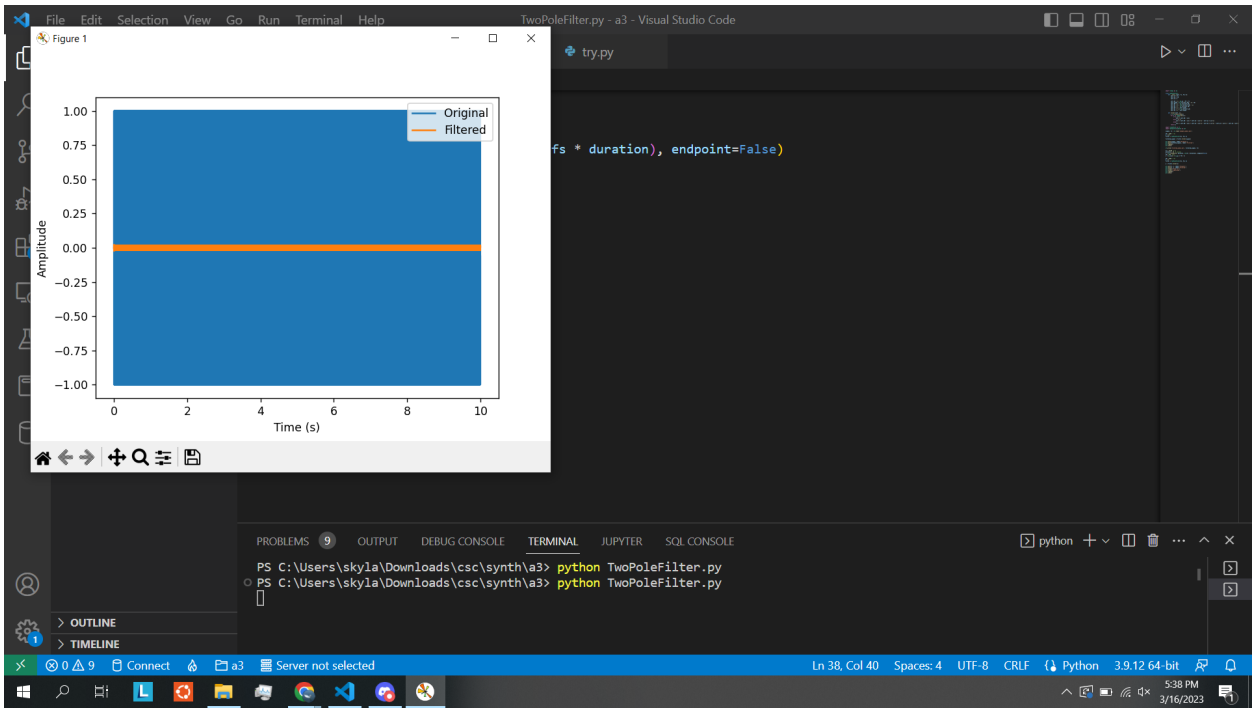
a noticeable attack in the sound that is outputted. Though it is noticeable it is still short for an attack.





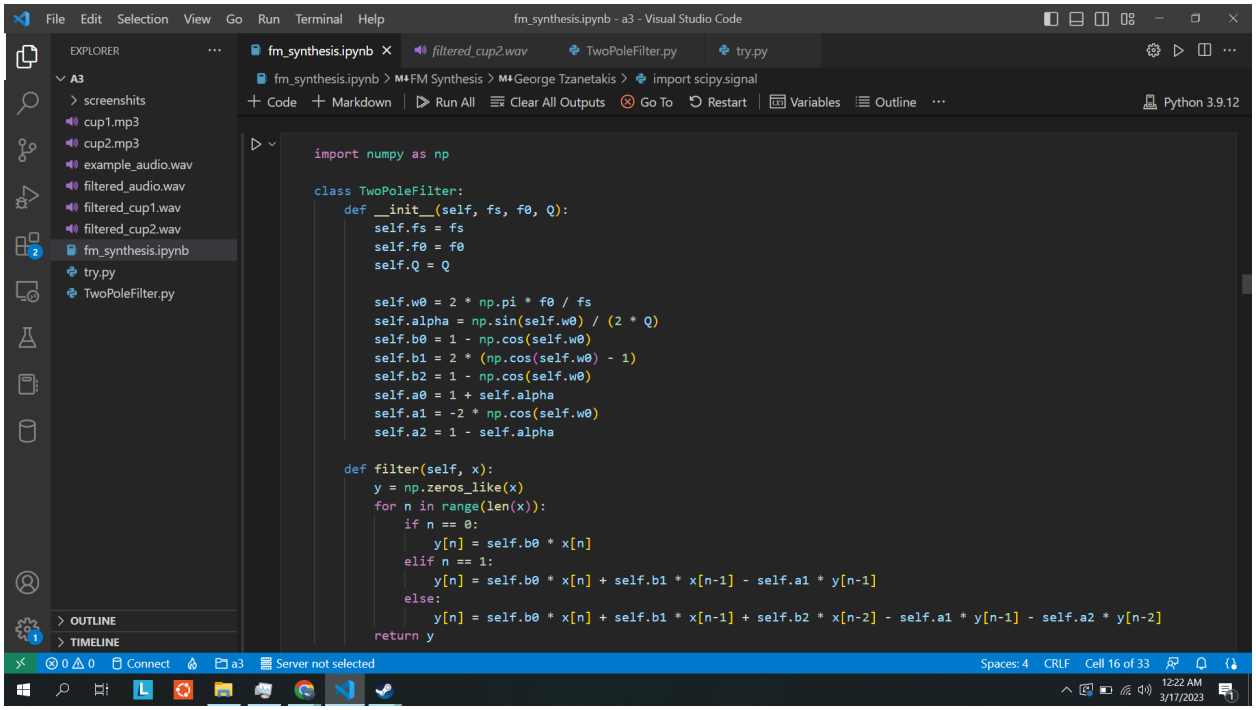

File Edit Selection View Go Run Terminal Help
TwoPoleFilter.py - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
TwoPoleFilter.py
try.py
TwoPoleFilter.py > ...
46
47 fs = 44100 # Hz
48 duration = 10 # seconds
49 t = np.linspace(0, duration, int(fs * duration), endpoint=False)
50 f0 = 440 # Hz
51 x = np.sin(2 * np.pi * f0 * t)
52
53 f0 = 1000 # Hz
54 Q = 5
55 filter = TwoPoleFilter(fs, f0, Q)
56
57 y = filter.filter(x)
58
59 plt.plot(t, x, label='Original')
60 plt.plot(t, y, label='Filtered')
61 plt.xlabel('Time (s)')
62 plt.ylabel('Amplitude')
63 plt.legend()
64 plt.show()
65
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER SQL CONSOLE
powershell
PS C:\Users\skyla\Downloads\csc\synth\a3> python TwoPoleFilter.py
PS C:\Users\skyla\Downloads\csc\synth\a3>
OUTLINE
TIMELINE
0 9 Connect a3 Server not selected
Ln 38, Col 40 Spaces: 4 UTF-8 CRLF Python 3.9.12 64-bit
5:37 PM
3/16/2023

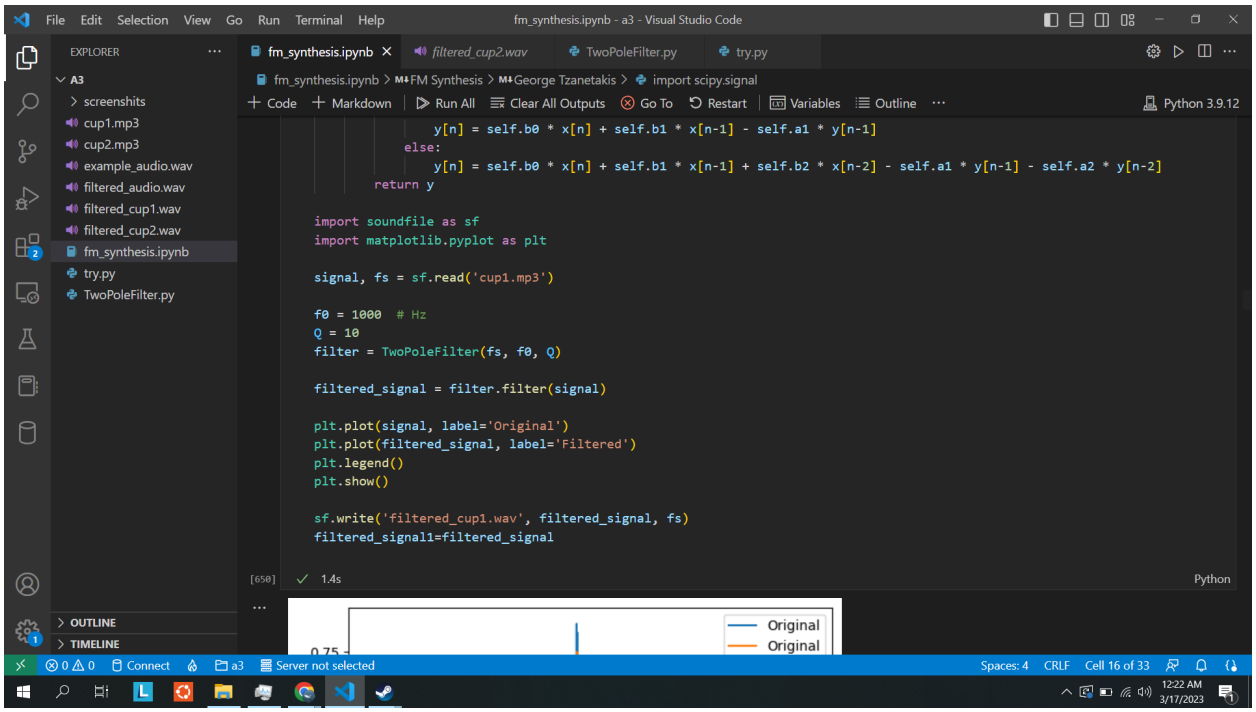


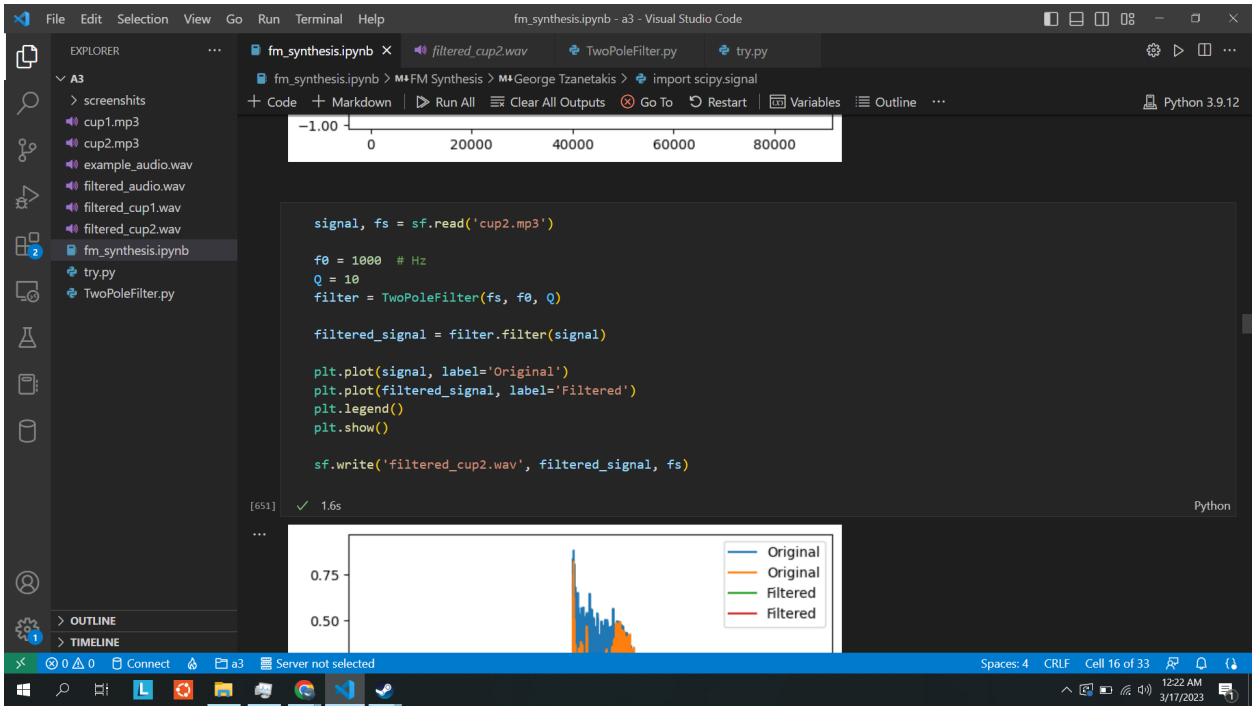


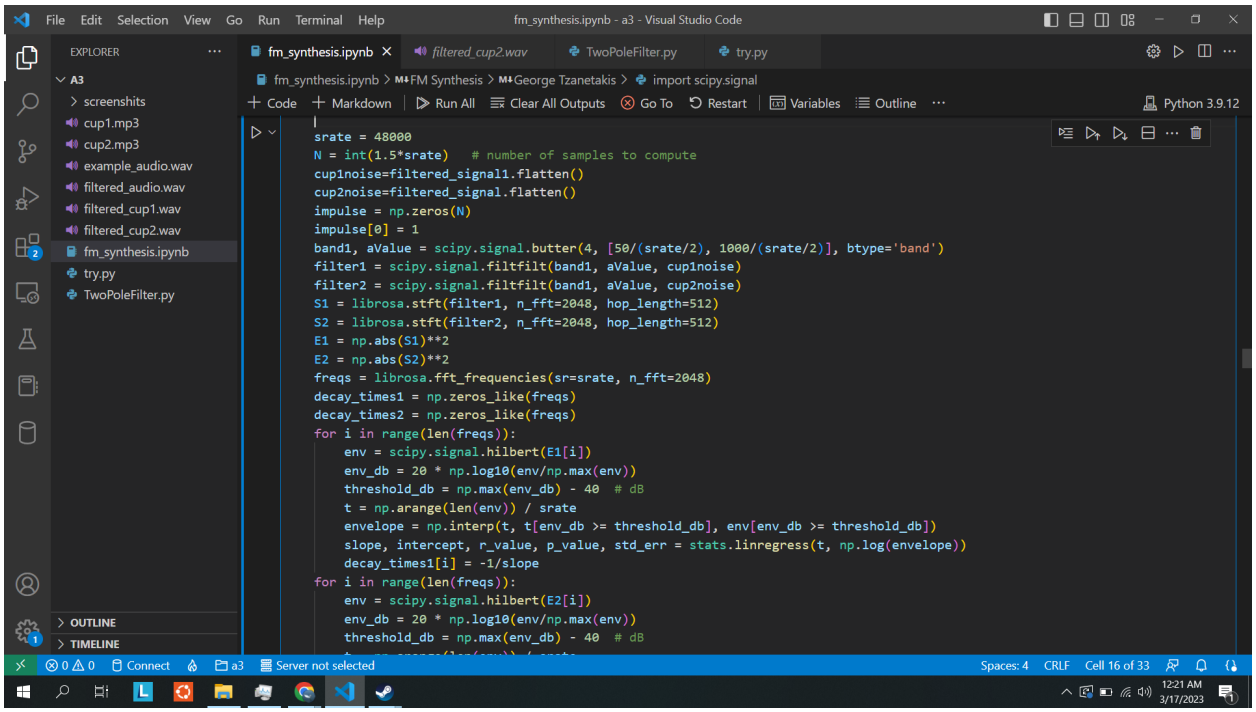
For question 8 we had to implement the parametric two pole filter used in the water bottle modal synthesis paper. Using this implementation we were to create two audio examples showing how it can be used. For my two audio examples I used different mediums, in the first example i used a one second wav file i got from a youtube video I found. The second example was of a basic sine wave. With both of the examples I ran them through the filter then compared the original sound wave to the filtered sound wave using a plotted graph. The filter itself is just based off of a collaboration of the information I had found in the paper, publication, and just online about the

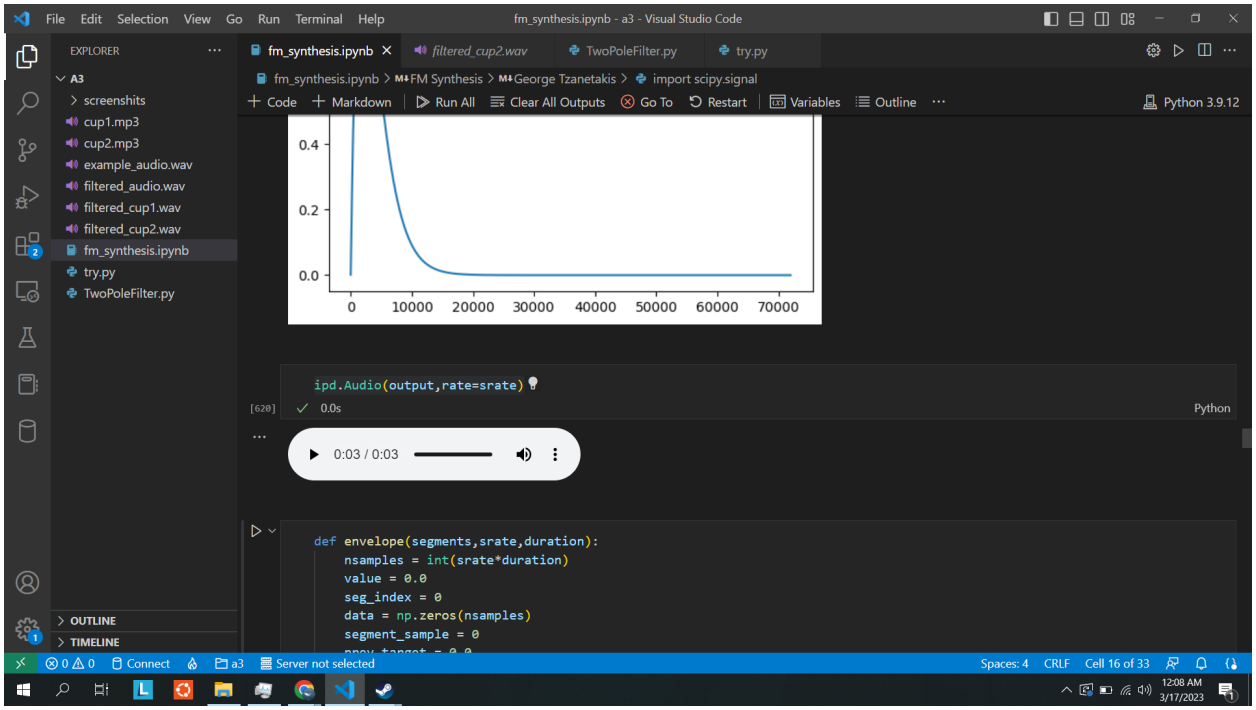
parametric two pole filter. There are three different formulas to use for the filter. The three conditions that trigger these different formulas are: 1. The first index, 2. The second index, 3. The rest of the indexes. Using this info you can use if else statements to check the correct index and give the correct formula.

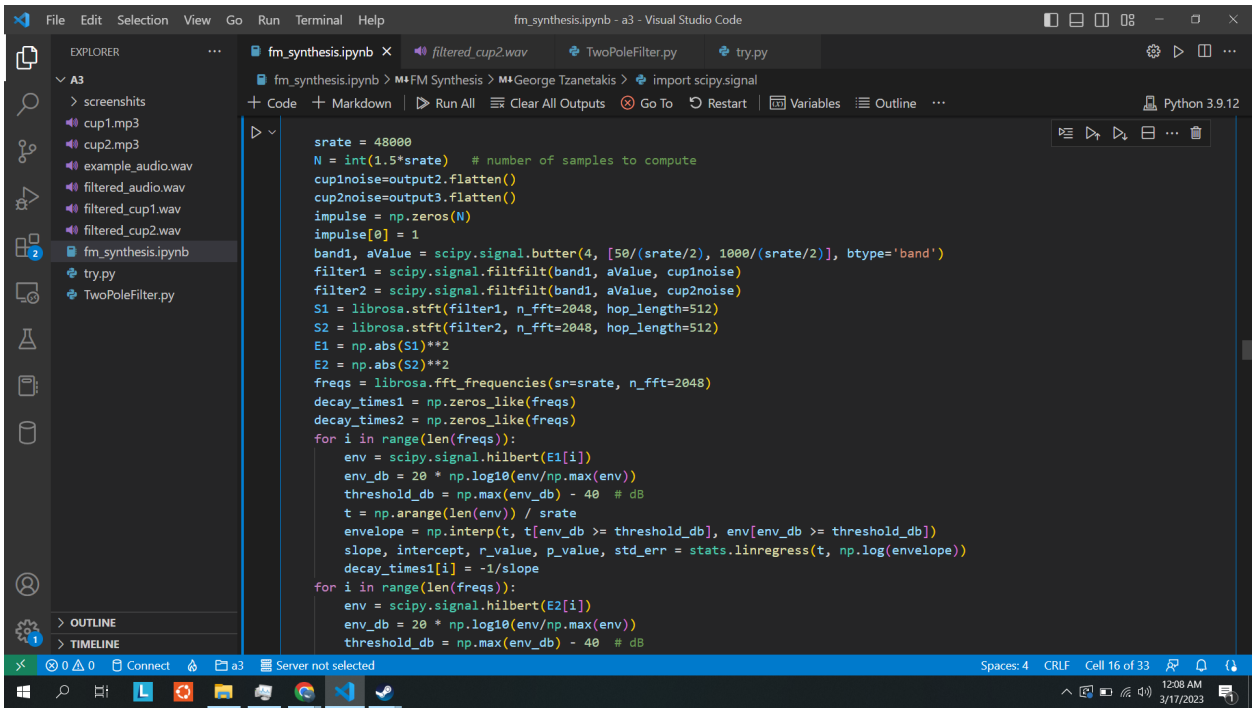

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
filtered_cup1.wav
filtered_cup2.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
filtered_cup2.wav
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > import scipy.signal
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
import numpy as np
class TwoPoleFilter:
def __init__(self, fs, f0, Q):
self.fs = fs
self.f0 = f0
self.Q = Q
self.w0 = 2 * np.pi * f0 / fs
self.alpha = np.sin(self.w0) / (2 * Q)
self.b0 = 1 - np.cos(self.w0)
self.b1 = 2 * (np.cos(self.w0) - 1)
self.b2 = 1 - np.cos(self.w0)
self.a0 = 1 + self.alpha
self.a1 = -2 * np.cos(self.w0)
self.a2 = 1 - self.alpha
def filter(self, x):
y = np.zeros_like(x)
for n in range(len(x)):
if n == 0:
y[n] = self.b0 * x[n]
elif n == 1:
y[n] = self.b0 * x[n] + self.b1 * x[n-1] - self.a1 * y[n-1]
else:
y[n] = self.b0 * x[n] + self.b1 * x[n-1] + self.b2 * x[n-2] - self.a1 * y[n-1] - self.a2 * y[n-2]
return y
OUTLINE
TIMELINE
0 0 Connect a3 Server not selected
Spaces: 4 CRLF Cell 16 of 33
12:22 AM 3/17/2023

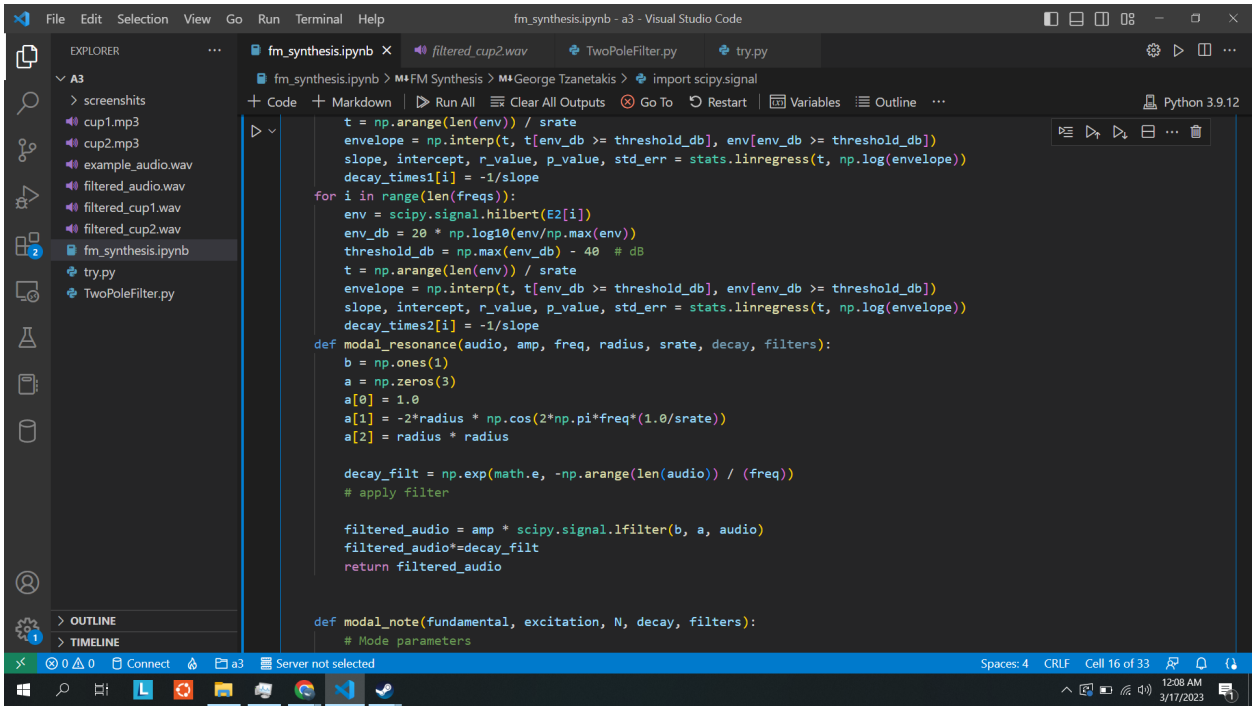

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
filtered_cup1.wav
filtered_cup2.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
filtered_cup2.wav
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > import scipy.signal
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
y[n] = self.b0 * x[n] + self.b1 * x[n-1] - self.a1 * y[n-1]
else:
y[n] = self.b0 * x[n] + self.b1 * x[n-1] + self.b2 * x[n-2] - self.a1 * y[n-1] - self.a2 * y[n-2]
return y
import soundfile as sf
import matplotlib.pyplot as plt
signal, fs = sf.read('cup1.mp3')
f0 = 1000 # Hz
Q = 10
filter = TwoPoleFilter(fs, f0, Q)
filtered_signal = filter.filter(signal)
plt.plot(signal, label='Original')
plt.plot(filtered_signal, label='Filtered')
plt.legend()
plt.show()
sf.write('filtered_cup1.wav', filtered_signal, fs)
filtered_signal1=filtered_signal
[650] 1.4s
Python
Original
Original
0.75
OUTLINE
TIMELINE
0 0 Connect a3 Server not selected
Spaces: 4 CRLF Cell 16 of 33
12:22 AM 3/17/2023


File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
filtered_cup1.wav
filtered_cup2.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
filtered_cup2.wav
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > import scipy.signal
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
−1.00
0
20000
40000
60000
80000
signal, fs = sf.read('cup2.mp3')
f0 = 1000 # Hz
Q = 10
filter = TwoPoleFilter(fs, f0, Q)
filtered_signal = filter.filter(signal)
plt.plot(signal, label='Original')
plt.plot(filtered_signal, label='Filtered')
plt.legend()
plt.show()
sf.write('filtered_cup2.wav', filtered_signal, fs)
[651]
1.6s
Python
0.75
0.50
Original
Original
Filtered
Filtered
OUTLINE
TIMELINE
Connect
a3
Server not selected
Spaces: 4
CRLF
Cell 16 of 33
12:22 AM
3/17/2023

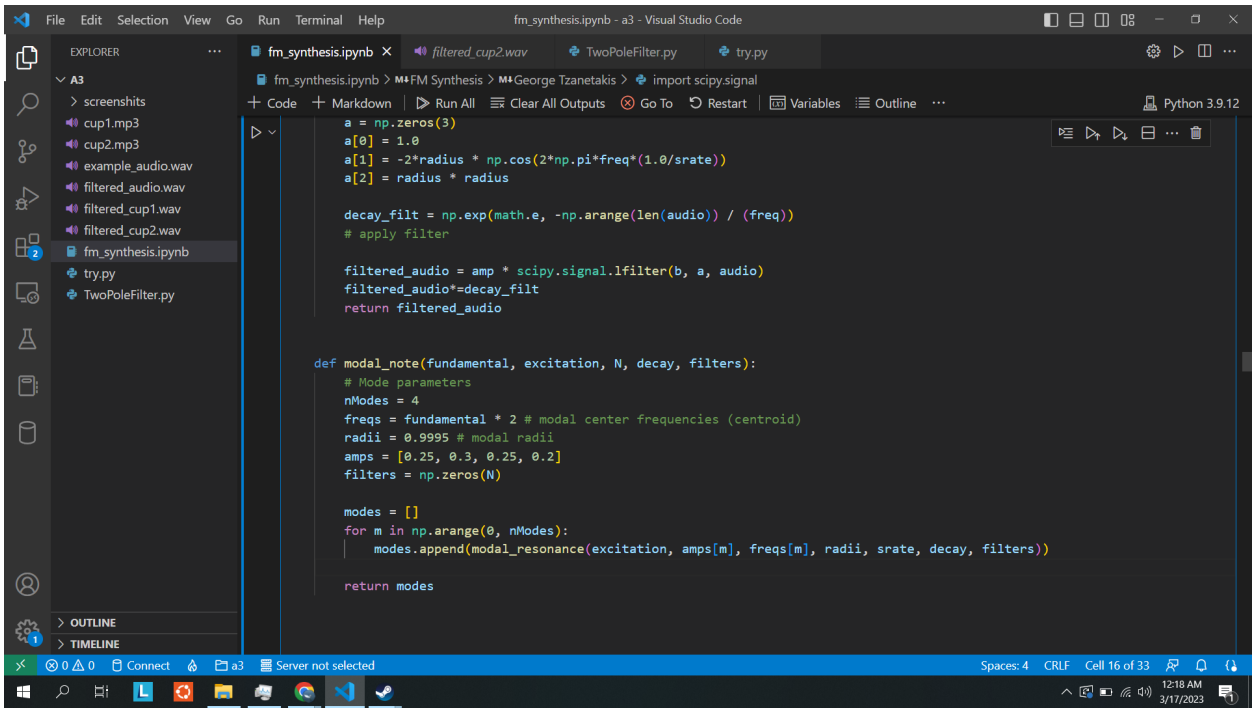

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
filtered_cup1.wav
filtered_cup2.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
filtered_cup2.wav
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > import scipy.signal
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
srate = 48000
N = int(1.5*srate) # number of samples to compute
cup1noise=filtered_signal1.flatten()
cup2noise=filtered_signal.flatten()
impulse = np.zeros(N)
impulse[0] = 1
band1, aValue = scipy.signal.butter(4, [50/(srate/2), 1000/(srate/2)], btype='band')
filter1 = scipy.signal.filtfilt(band1, aValue, cup1noise)
filter2 = scipy.signal.filtfilt(band1, aValue, cup2noise)
S1 = librosa.stft(filter1, n_fft=2048, hop_length=512)
S2 = librosa.stft(filter2, n_fft=2048, hop_length=512)
E1 = np.abs(S1)**2
E2 = np.abs(S2)**2
freqs = librosa.fft_frequencies(sr=srate, n_fft=2048)
decay_times1 = np.zeros_like(freqs)
decay_times2 = np.zeros_like(freqs)
for i in range(len(freqs)):
env = scipy.signal.hilbert(E1[i])
env_db = 20 * np.log10(env/np.max(env))
threshold_db = np.max(env_db) - 40 # dB
t = np.arange(len(env)) / srate
envelope = np.interp(t, t[env_db >= threshold_db], env[env_db >= threshold_db])
slope, intercept, r_value, p_value, std_err = stats.linregress(t, np.log(envelope))
decay_times1[i] = -1/slope
for i in range(len(freqs)):
env = scipy.signal.hilbert(E2[i])
env_db = 20 * np.log10(env/np.max(env))
threshold_db = np.max(env_db) - 40 # dB
OUTLINE
TIMELINE
Connect
a3
Server not selected
Spaces: 4
CRLF
Cell 16 of 33
12:21 AM
3/17/2023


fm_synthesis.ipynb - a3 - Visual Studio Code
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000
ipd.Audio(output,rate=srate)
0.0s
Python
0:03 / 0:03
def envelope(segments,srate,duration):
nsamples = int(srate*duration)
value = 0.0
seg_index = 0
data = np.zeros(nsamples)
segment_sample = 0

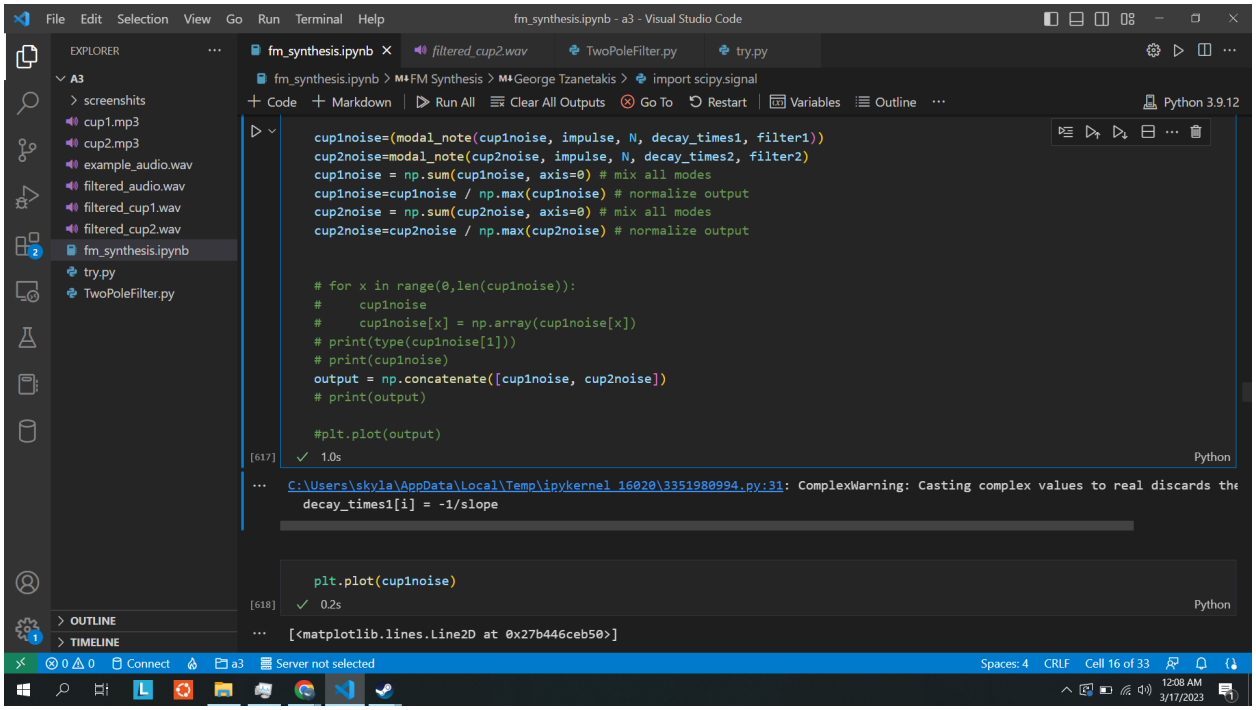

fm_synthesis.ipynb - a3 - Visual Studio Code
srate = 48000
N = int(1.5*srate) # number of samples to compute
cup1noise=output2.flatten()
cup2noise=output3.flatten()
impulse = np.zeros(N)
impulse[0] = 1
band1, aValue = scipy.signal.butter(4, [50/(srate/2), 1000/(srate/2)], btype='band')
filter1 = scipy.signal.filtfilt(band1, aValue, cup1noise)
filter2 = scipy.signal.filtfilt(band1, aValue, cup2noise)
S1 = librosa.stft(filter1, n_fft=2048, hop_length=512)
S2 = librosa.stft(filter2, n_fft=2048, hop_length=512)
E1 = np.abs(S1)**2
E2 = np.abs(S2)**2
freqs = librosa.fft_frequencies(sr=srate, n_fft=2048)
decay_times1 = np.zeros_like(freqs)
decay_times2 = np.zeros_like(freqs)
for i in range(len(freqs)):
env = scipy.signal.hilbert(E1[i])
env_db = 20 * np.log10(env/np.max(env))
threshold_db = np.max(env_db) - 40 # dB
t = np.arange(len(env)) / srate
envelope = np.interp(t, t[env_db >= threshold_db], env[env_db >= threshold_db])
slope, intercept, r_value, p_value, std_err = stats.linregress(t, np.log(envelope))
decay_times1[i] = -1/slope
for i in range(len(freqs)):
env = scipy.signal.hilbert(E2[i])
env_db = 20 * np.log10(env/np.max(env))
threshold_db = np.max(env_db) - 40 # dB

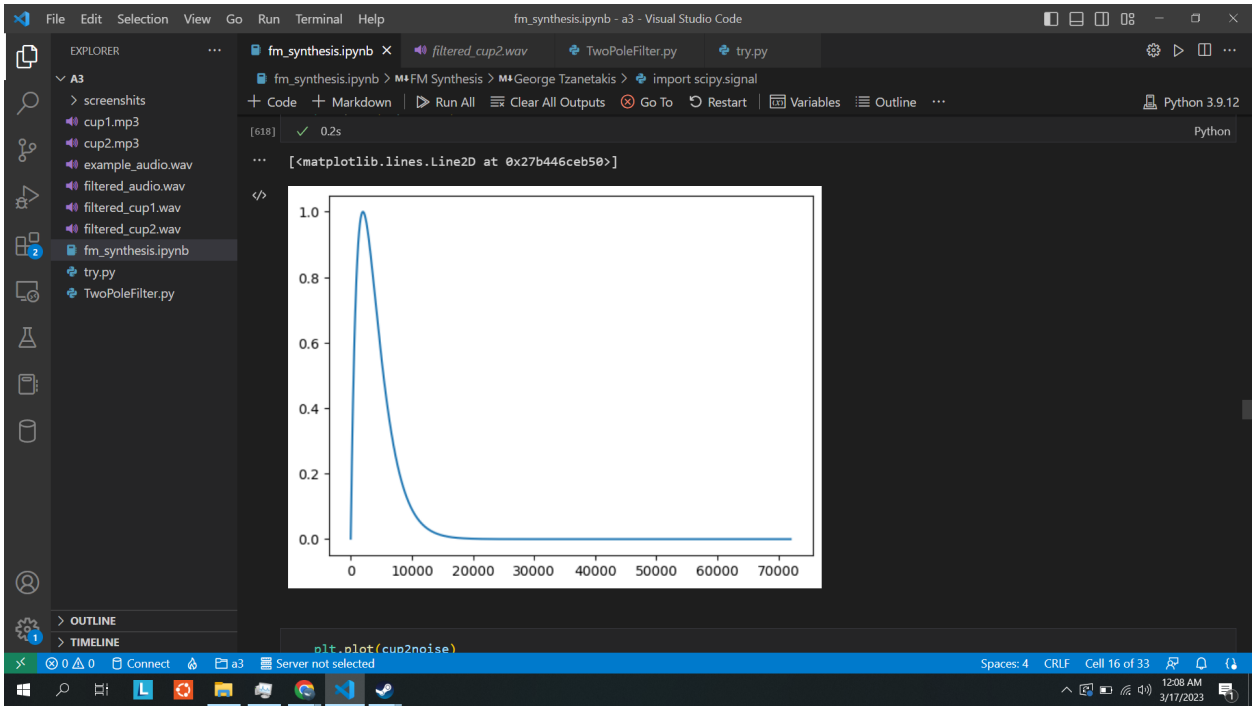


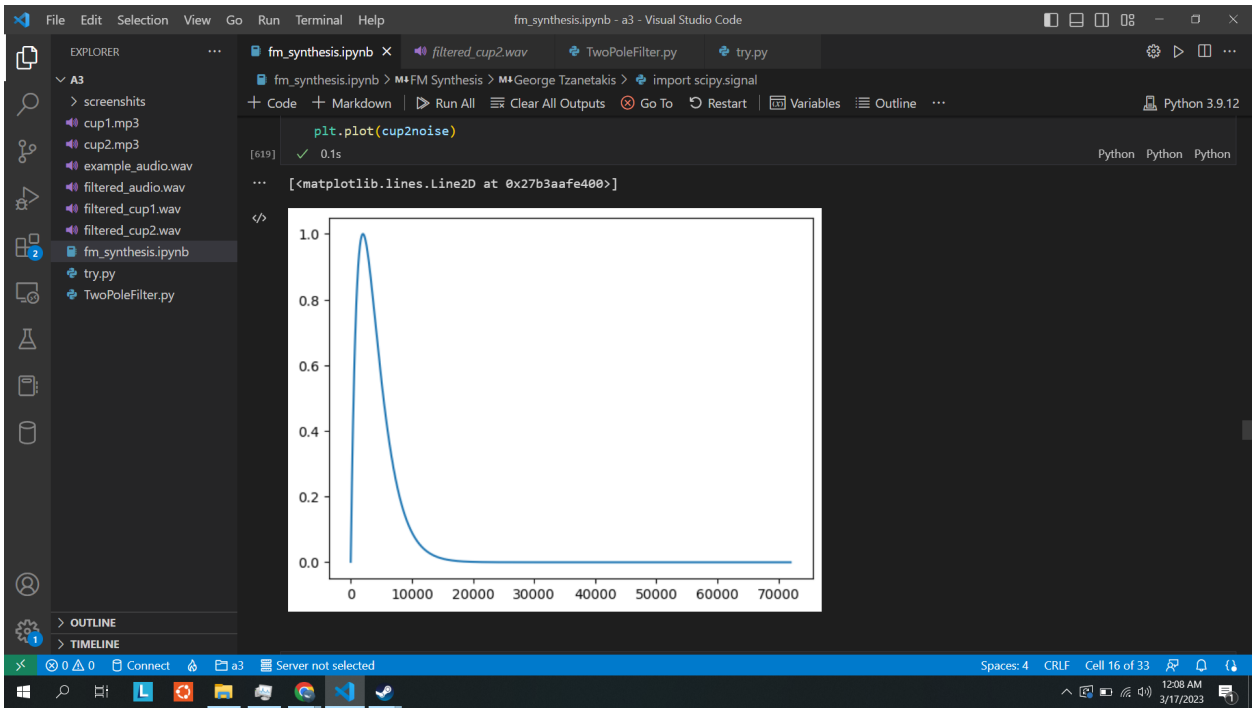
```
        t = np.arange(len(env)) / srate
        envelope = np.interp(t, t[env_db >= threshold_db], env[env_db >= threshold_db])
        slope, intercept, r_value, p_value, std_err = stats.linregress(t, np.log(envelope))
        decay_times1[i] = -1/slope
    for i in range(len(freqs)):
        env = scipy.signal.hilbert(E2[i])
        env_db = 20 * np.log10(env/np.max(env))
        threshold_db = np.max(env_db) - 40  # dB
        t = np.arange(len(env)) / srate
        envelope = np.interp(t, t[env_db >= threshold_db], env[env_db >= threshold_db])
        slope, intercept, r_value, p_value, std_err = stats.linregress(t, np.log(envelope))
        decay_times2[i] = -1/slope
    def modal_resonance(audio, amp, freq, radius, srate, decay, filters):
        b = np.ones(1)
        a = np.zeros(3)
        a[0] = 1.0
        a[1] = -2*radius * np.cos(2*np.pi*freq*(1.0/srate))
        a[2] = radius * radius

        decay_filt = np.exp(math.e, -np.arange(len(audio)) / (freq))
        # apply filter

        filtered_audio = amp * scipy.signal.lfilter(b, a, audio)
        filtered_audio*=decay_filt
        return filtered_audio


    def modal_note(fundamental, excitation, N, decay, filters):
        # Mode parameters
```



```
        a = np.zeros(3)
        a[0] = 1.0
        a[1] = -2*radius * np.cos(2*np.pi*freq*(1.0/srate))
        a[2] = radius * radius

        decay_filt = np.exp(math.e, -np.arange(len(audio)) / (freq))
        # apply filter

        filtered_audio = amp * scipy.signal.lfilter(b, a, audio)
        filtered_audio*=decay_filt
        return filtered_audio


    def modal_note(fundamental, excitation, N, decay, filters):
        # Mode parameters
        nModes = 4
        freqs = fundamental * 2 # modal center frequencies (centroid)
        radii = 0.9995 # modal radii
        amps = [0.25, 0.3, 0.25, 0.2]
        filters = np.zeros(N)

        modes = []
        for m in np.arange(0, nModes):
            modes.append(modal_resonance(excitation, amps[m], freqs[m], radii, srate, decay, filters))

        return modes
```

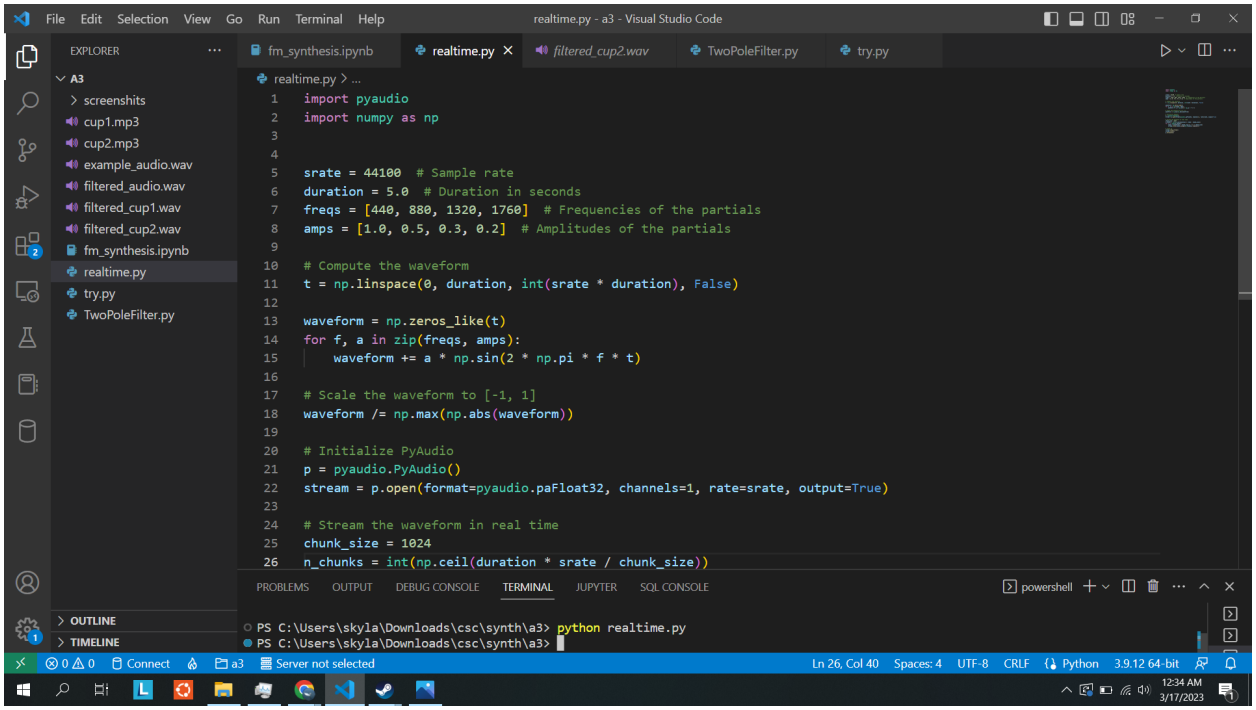

File Edit Selection View Go Run Terminal Help
fm_synthesis.ipynb - a3 - Visual Studio Code
EXPLORER
A3
screenshits
cup1.mp3
cup2.mp3
example_audio.wav
filtered_audio.wav
filtered_cup1.wav
filtered_cup2.wav
fm_synthesis.ipynb
try.py
TwoPoleFilter.py
fm_synthesis.ipynb
filtered_cup2.wav
TwoPoleFilter.py
try.py
fm_synthesis.ipynb > FM Synthesis > George Tzanetakis > import scipy.signal
Code Markdown Run All Clear All Outputs Go To Restart Variables Outline
Python 3.9.12
cup1noise=(modal_note(cup1noise, impulse, N, decay_times1, filter1))
cup2noise=modal_note(cup2noise, impulse, N, decay_times2, filter2)
cup1noise = np.sum(cup1noise, axis=0) # mix all modes
cup1noise=cup1noise / np.max(cup1noise) # normalize output
cup2noise = np.sum(cup2noise, axis=0) # mix all modes
cup2noise=cup2noise / np.max(cup2noise) # normalize output
# for x in range(0,len(cup1noise)):
# cup1noise
# cup1noise[x] = np.array(cup1noise[x])
# print(type(cup1noise[1]))
# print(cup1noise)
output = np.concatenate([cup1noise, cup2noise])
# print(output)
#plt.plot(output)
[617] 1.0s
Python
C:\Users\skyla\AppData\Local\Temp\ipykernel_16020\3351980994.py:31: ComplexWarning: Casting complex values to real discards the
decay_times1[i] = -1/slope
plt.plot(cup1noise)
[618] 0.2s
Python
[<matplotlib.lines.Line2D at 0x27b446ceb50>]
OUTLINE
TIMELINE
0 0 Connect a3 Server not selected
Spaces: 4 CRLF Cell 16 of 33
12:08 AM
3/17/2023

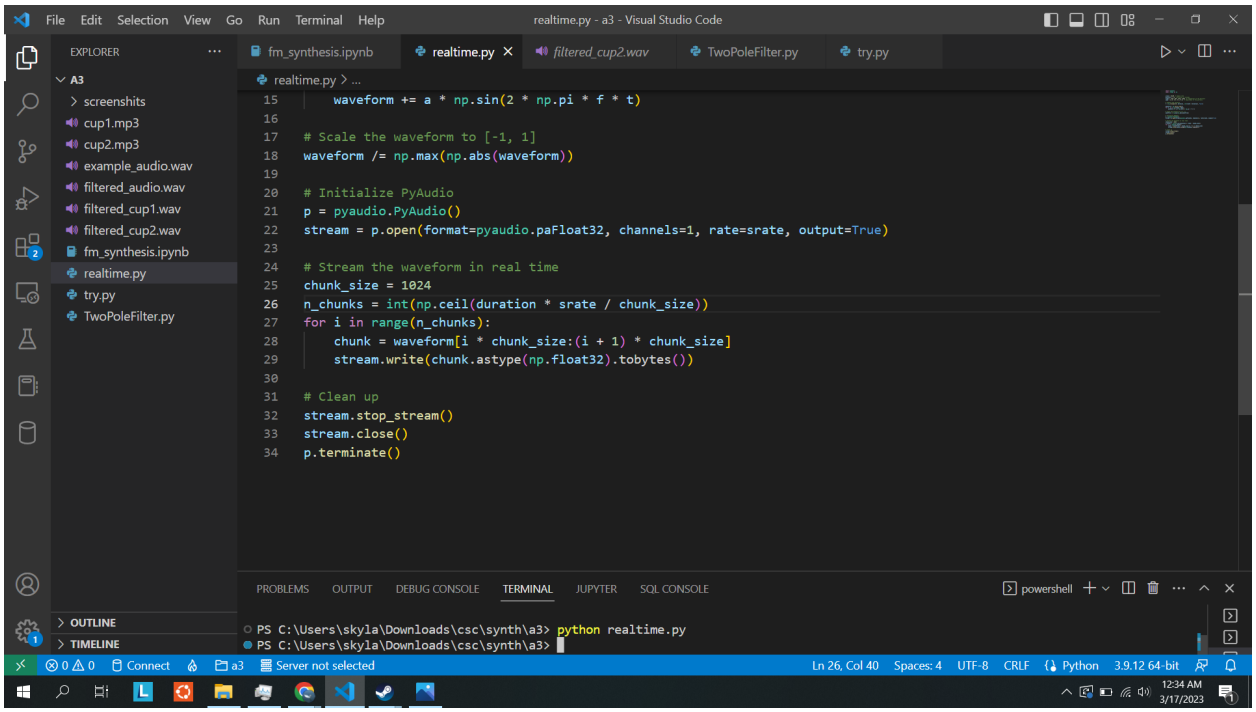




For Question 9 we had to use the parametric two-pole filter to create a modal synthesis approximation for the two coffee mug noises.I used the process for estimating the mode decay that was shown in the water bottle modal synthesis paper for this question. The paper describes a process for modeling the decay behavior of an object at different frequencies. I used the cup's sound data to estimate the decay rate for each mode, and then generated a filter that simulates the same sound. I used this filter to make a synthesized sound. The output allows for realistic sound synthesis. I had to get all of the variables and such twice for each of the cup files but I

was able to run them both through the functions so that saved me a little bit of effort and time. I was able to use code from both previous questions involving modal synthesis and the two-pole filter. The main problem I was having was making sense of the “WATER BOTTLE SYNTHESIS WITH MODAL SIGNAL PROCESSING” article. It is always difficult for me to put research papers in a way that I can understand it and use the information from it in a practical manner. But eventually I started putting the pieces together and saw how different formulas fit together to get the end result and the delay array.





For question 10 I had to implement a real-time version of any of the synthesis models you developed in this assignment. I could have chosen to use any language/programming environment for this question. I chose to make a program in python to make a real time sound with additive synthesis. The program makes a sound by adding together four different kinds of sound waves. Each sound wave has a different pitch and loudness. The program then plays the sound through the speakers of your computer in real-time. I decided to use pyaudio for this

program because pyaudio allows for real time sound playing. Most of the code and information I needed I got from the previous question I had done on additive synthesis.