Assignment

Assignment A2

Alexandre LAGARRUE

A T-717-SPST Assignment

April 27, 2022

1.1 - Create 4 frequency components, all sampled at ‘16,000Hz‘ for 5 seconds. These components should have the frequencies ‘500‘, ‘1000‘, ‘2000‘, ‘5000‘Hz. The relative amplitude scalar values should be ‘1‘, ‘0.5‘, ‘0.1‘, ‘0.05‘. Save the audio and play it back. This combination should sound distinctly different from any of the singular pure tone sine waves it consists of.

Listen to multisine.wav .

2- Create the same weighted sum as in 1.1 and use ‘tools.plot_spectrum() to show the spectrum. Describe what you see. Can you identify which values in the frequency domain belong to which component in time domain?

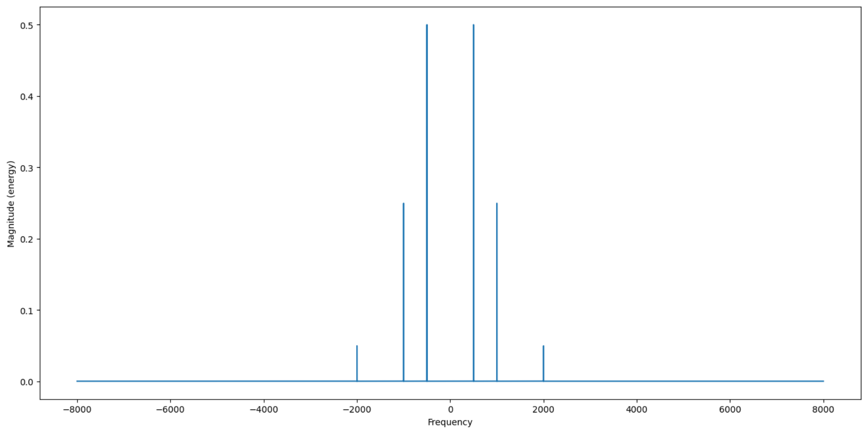

Figure 1:

We can see the components for 500Hz, 1000Hz and 2000Hz with their respective amplitudes. However, we can't see the component for 5000Hz because its amplitude is too low (0.05).

3 - Use the ‘tools.plot_spectrogram()‘ function to plot the Mel-scale spectrogram for the combination you created earlier. plot the Mel-scale spectrogram for ‘./data/f1.wav‘. Given the information above, can you explain why they are so different?

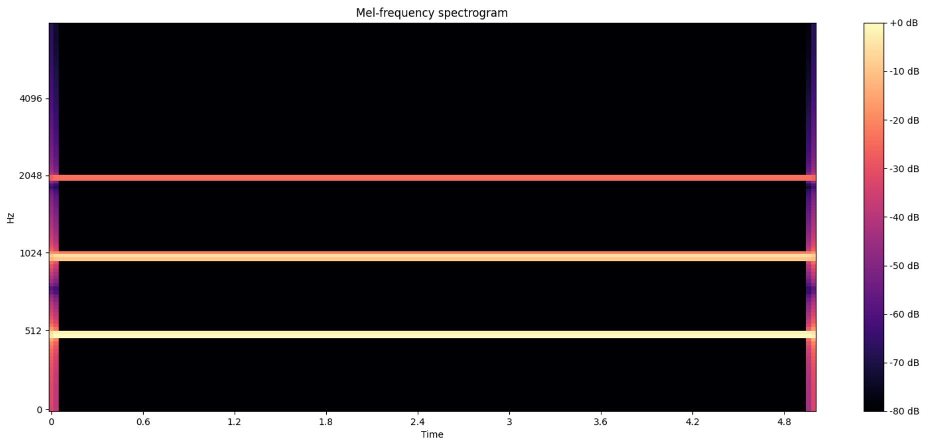

Figure 2:

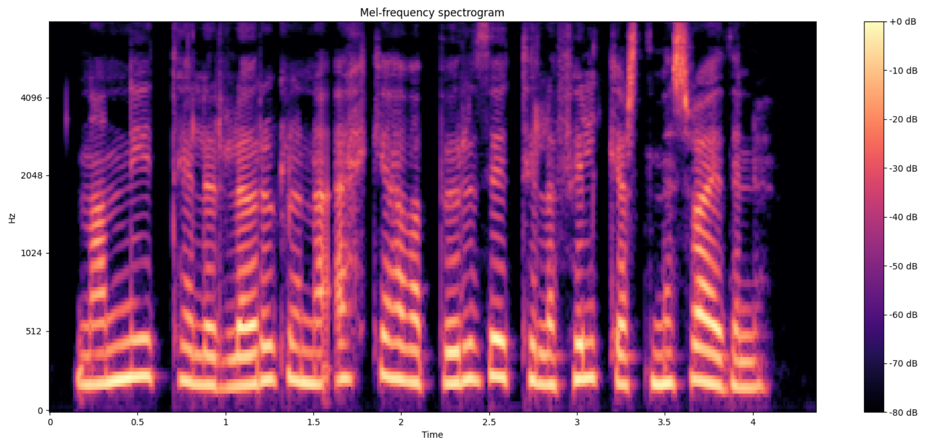

Figure 3:

We can explain the difference by the fact that the multi sine is composed of 4 frequency components (500, 1000, 2000, 5000Hz) while f1.wav is a recording of a speech which is composed of much more frequencies. Also, the tone is flat for the multi sine with no variation.

2.1 - Load a sample from the ‘./data‘ directory and half the number of sample using your function. Save the waveform and keep the same sample rate. Describe the results. Why does this happen?

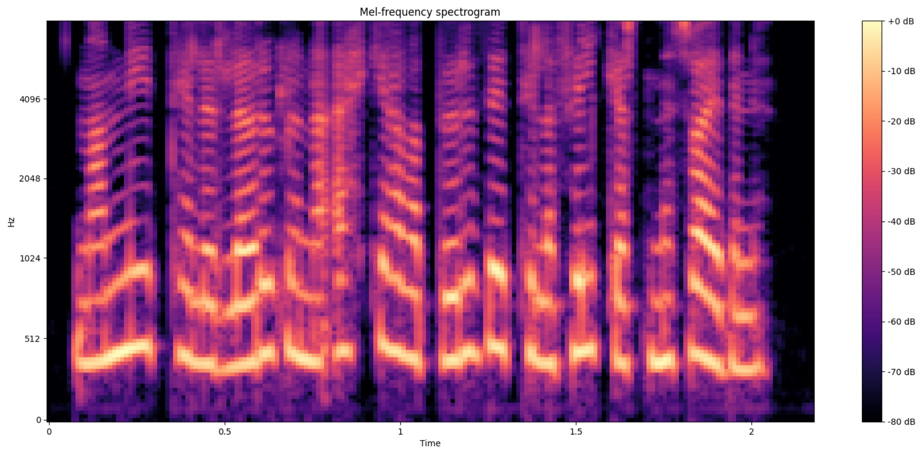

Figure 4:

I loaded the sample f1.wav and saved the waveform with half the samples as half_sr.wav in ./data/ . The result is the sample being two times faster and higher in tone too. This happens because we have two times less samples but with the same sample rate.

2.2 - Now do the same but half the sample rate as well when you save the waveform. Describe the results. Why does this happen?

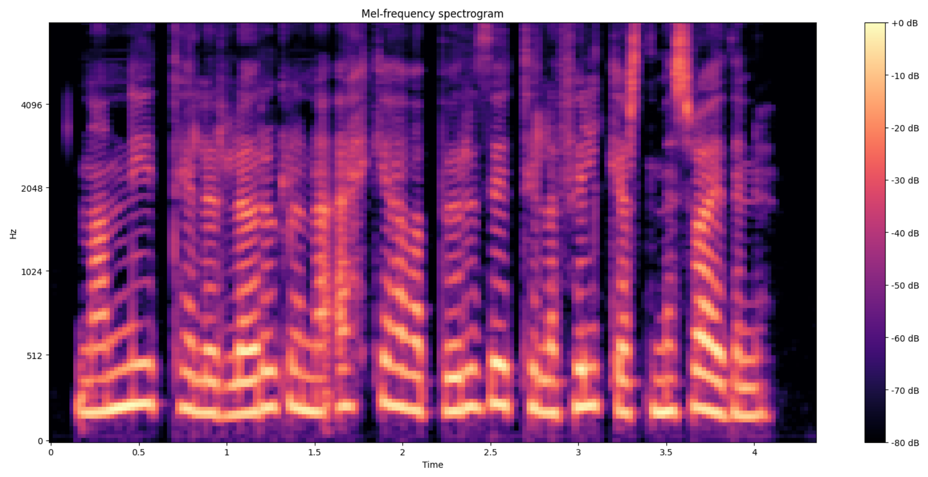

Figure 5:

This new waveform is saved as half_sr2.wav in ./data/ . After doing the same and halving the sample rate, the sound now sounds normal. Because we divided both the number of samples and the sample rate by 2, the waveform is proportional to the initial one.

2.3 - Next repeat step two but remove ever more samples while reducing the sample rate at an equal rate. Compare the output for sr, sr/2, sr/4, sr/8, sr/16. sr/32. At what point does the audio get too distorted in your opinion? At what point could you no longer understand what is being said. Calculate what the actual sample rate is at these points and compare to what is used in telephone audio (8000 Hz).

All the waveforms described above are respectively saved as sr_over_2_m2.wav, sr_over_4_m2.wav, sr_over_8_m2.wav, sr_over_16_m2.wav, sr_over_32_m2.wav .

Up to sr/8, it sounds completely normal, although we can observe a loss of quality at sr/8. At sr/16 and sr/32, there isn't any sound. This might be because we reach frequencies out of the hearing range. So the last sample rate where we perceive results is sr/8, or 5512Hz, which is lower than the sample rate in telephone audio.

3.1 - Create a function ‘zip_waves(a, b)‘ that takes in two waveforms and zips them together (e.g. a1-b1-a2-b2-a3-...). Load two samples of your choice from ‘./data‘. Mix them by using your zip function. Note: The samples in ‘./data‘ are all of different lengths so cut the samples like shown in ‘./example.py‘. Plot the results. Save the waveform and describe what you hear.

I loaded f1.wav and f2.wav and ziped them together.

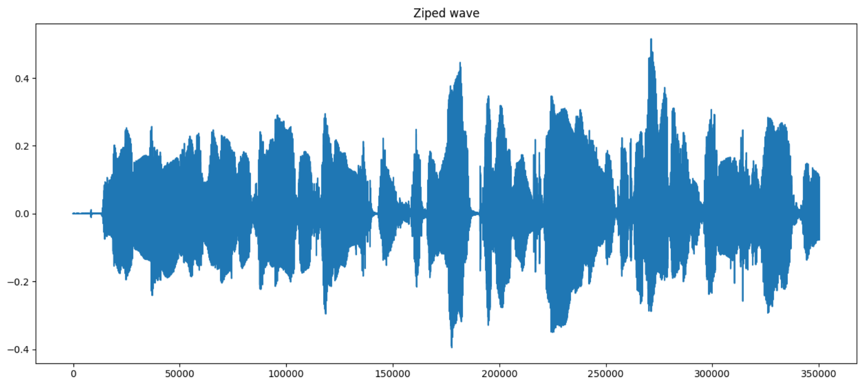

Figure 6:

The zipped waveform is saved as ziped.wav in ./data/ . When listening to the waveform, we can hear the two samples overlapping, two slower and at a lower tone.

3.2 - What is needed to do get the audio to sound more normal? Produce a normal sounding mixed waveform by changing the sample rate. Compare this result by mixing the waveforms like you do in your ‘multi_sine‘ function. Create a function ‘mix_waves(a, b, c_a, c_b)‘ where ‘a‘ and ‘b‘ are waveforms and ‘c_a‘ and ‘c_b‘ are the relative amplitude scalars for each. Each sample from combining waveforms samples ‘a_i‘ and ‘b_i‘ in this way should be ‘c_a*a_i + c_b*b_i‘. Choose whatever ‘c‘ values you like the most! Do you hear a difference between these two results? Why / Why not?

To get the audio to sound more normal, we need to multiply the sample rate by a factor of 2. I saved the result as ziped_normal.wav in ./data/ . I mixed together the waveforms f1.wav and f2.wav, giving f1.wav and amplitude scalar of 4 and f2.wav an amplitude scalar of 0.25 . I saved the result as mixed.wav in ./data/ .

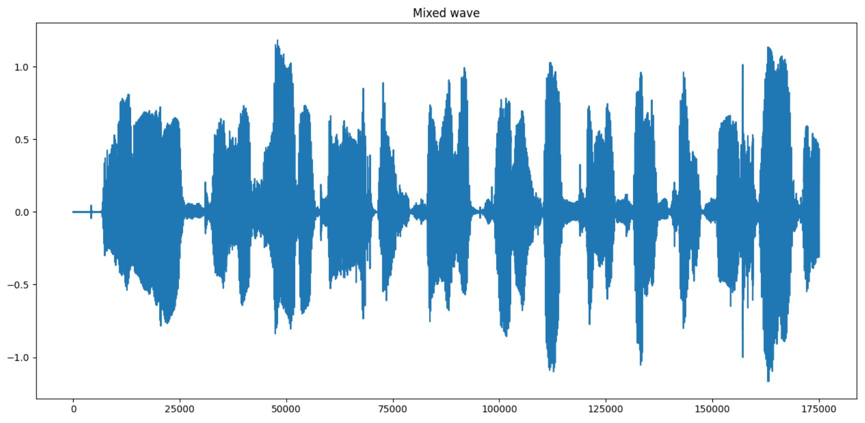

Figure 7:

The difference between these two results is that in the mixed waveform, one sample on louder than the other one, due to the amplitude scalar that I chose. If I chose 1 as the amplitude scalar for both f1.wav and f2.wav, it would sound exactly the same as the zipped waveforms with the previous change.

3.3 - Create a function ‘fade(wave, start_val, stop_val)‘ that fades a waveform such that the first sample has a scaling factor of ‘start_val‘ and the last has a scaling factor of ‘stop_val‘ where the scales increase linearly from start to finish (use ‘np.linspace()‘ for this). Apply this type of fade on any sample from ‘./data‘ with ‘start_val=0‘ and ‘stop_val=1‘. Plot the resulting waveform. Listen to the output. Describe the results.

I loaded the sample m2.wav. I saved the result as fade.wav in ./data/ .

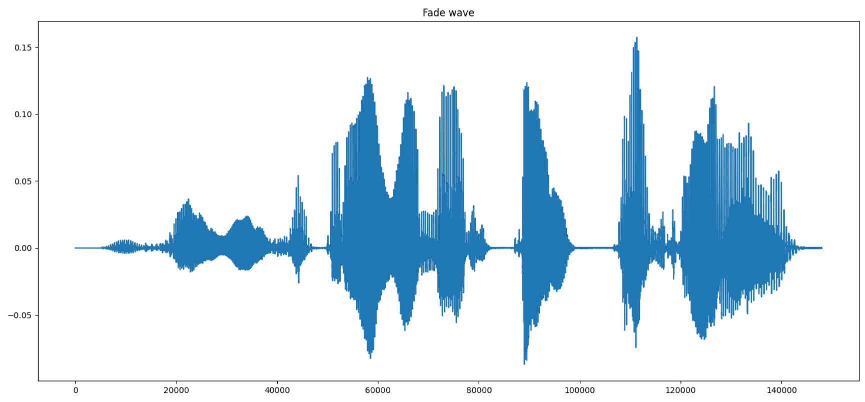

Figure 8:

When listening to the output, We notice that when the recording starts it’s very low, then it become gradually louder until it reaches its normal loudness at the end.

4 - Create something! You can do whatever you like here using the methods we have uncovered already and methods used in ‘example.py‘. Explain what you decided to do

This is what i decided to do :

- Open f1.wav and m2.wav
- Take the first 0 to 3 seconds of f1.wav and m2.wav
- Mix the segments together, with a coefficient of 2 for f1.wav and 0.25 for m2.wav
- Same step as 3. but with a coefficient of 0.25 for f1.wav and 2 for m2.wav
- Take the same segments from f1.wav and m2.wav and zip them together while using half the sample rate so the sound is not distorded
- Fade f1.wav from 0 to 2 and m2.wav from 2 to 0
- Concatenate the different waves together to be able to hear each transformation separatly
- Save the result as combination_f1m2.wav in ./data/

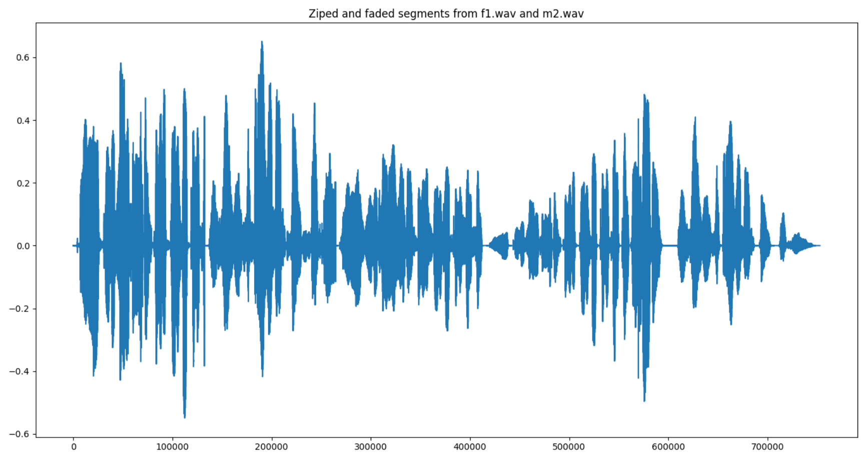

Figure 9:

At the beginning we can visualize the two segment mixed with a max amplitude of 2, then the zipped segments in the middle and finally the two segment respectively becoming louder and less loud.