EC601 Mini Project 3

DSP Visualization By Python

-- Ganghao Li

DSP introduction

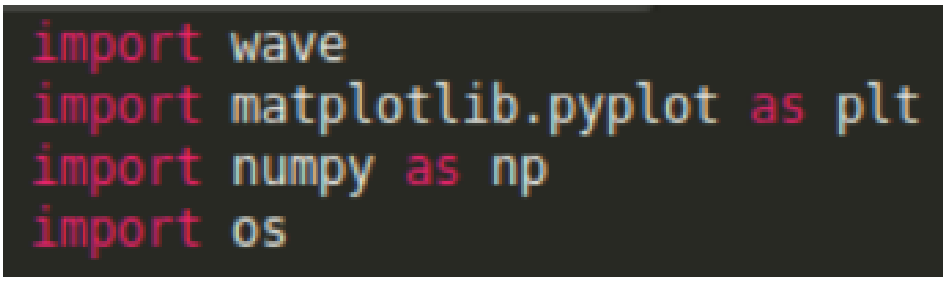
Digital signal processing is the theory and technique of digitally representing and processing signals. Digital signal processing and analog signal processing are a subset of signal processing. Mainly used in high-speed mathematical operations and real-time processing of data, voice, and video signals

Therefore, before performing digital signal processing, the signal needs to be converted from the analog domain to the digital domain, which is usually achieved by an analog-to-digital converter. And the output of digital signal processing often has to be transformed into the analog domain, which is achieved by a digital-to-analog converter

But, when people working on DSP, they find it is hard to see a large amount of results of abstract function, so they use some visualization tools to promote it.

Method Background

I realize the visualization function by using Python, so I find some library and method in python to solve this problem, which are wave, numpy and matplotlib.



```
import wave
import matplotlib.pyplot as plt
import numpy as np
import os
```

1. wave

The program can read the file as ‘.txt’ which stored the information from digital signal. But under this method, which request user to transform the digital signal to string in the text. For example, we should first find each element’s value and time in digital signal source, which is relatively difficult to deal with.

So I choose the input file as ‘.wav’, many of the songs’ format, which is more usual in our daily life and we do not need to take more process on this kind of file. The object ’wave’ can be processed by python directly and it concluded many methods like get the parameters and forms from input file.

2. numpy

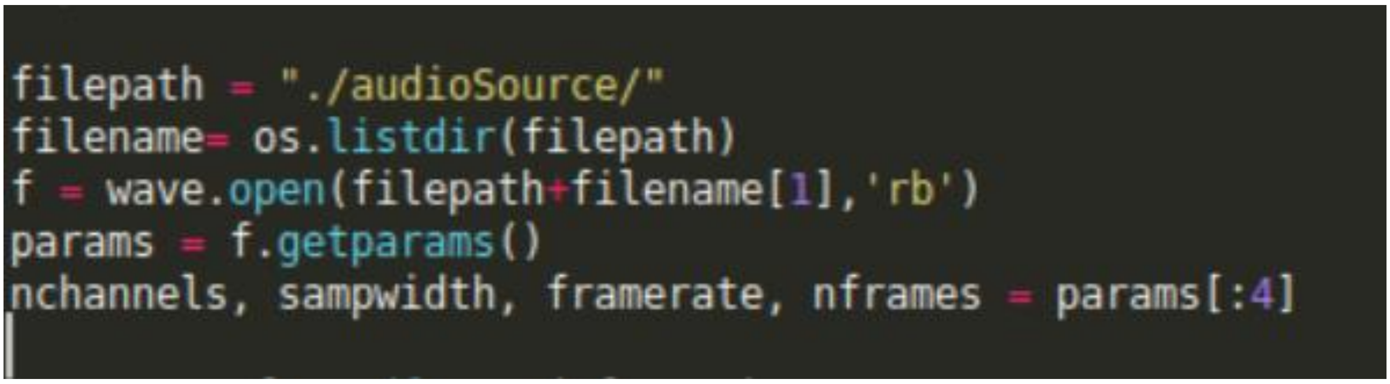
The reason to utilize numpy is that I need to transfer the data in the ‘.wav’ from ‘string’ to ‘int’ like ‘frombuffer()’ and use some math methods which can realize the math function more easily.

3. matplotlib

This is the key method in this solution, because with this method, we can realize the visualization, to decide whether need the grid, to set the xLabel, yLabel, title and something else.

Solution

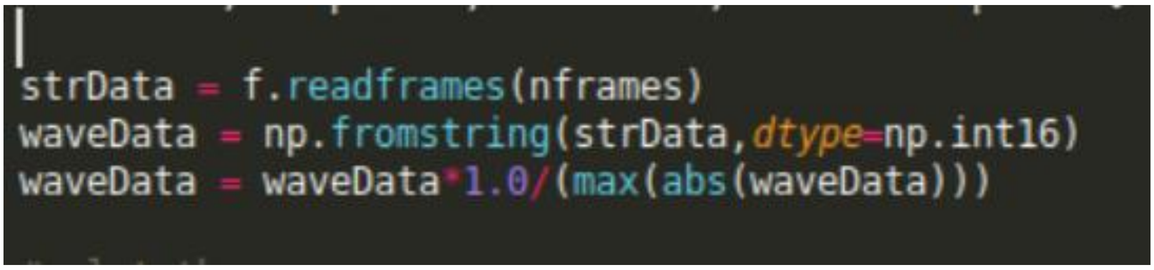
1. Read the ‘.wav’ file and get the waves’ parameters object. Then we extract this object and obtain the first four parameters, which are n channels, sample width, frame rate, n frames.



```
filepath = "./audioSource/"
filename= os.listdir(filepath)
f = wave.open(filepath+filename[1],'rb')
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
```

2. Read the audio file’s frames, return the string result and transform it to ‘int’ format. Notice that here we’d better use numpy.frombuffer() instead of np.fromstring, otherwise the console will show that Deprecation Warning: That means yhe binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs.

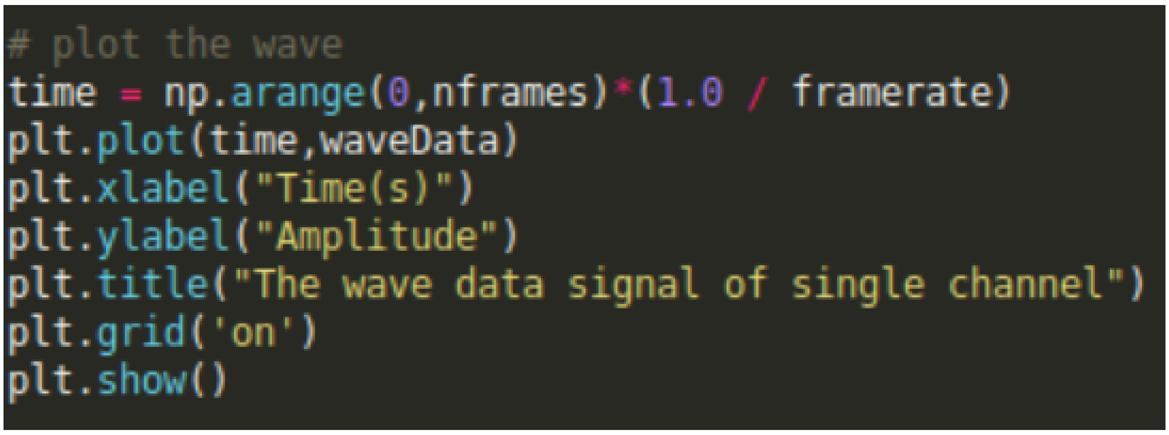
3. Normalize the amplitude of the ‘wave’, because then we can see the data in a standard way.



```
strData = f.readframes(nframes)
waveData = np.fromstring(strData,dtype=np.int16)
waveData = waveData*1.0/(max(abs(waveData)))
```

4. use numpy.arrange() to set the time

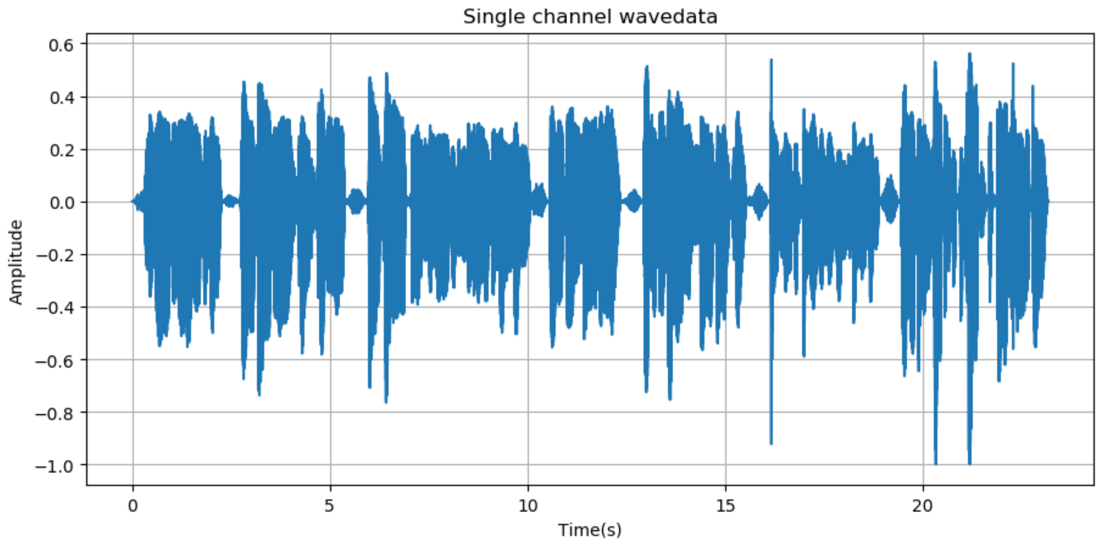
5. Start to plot. I add the plot, xLable, yLable, title and grid method to make the photo more beautiful.



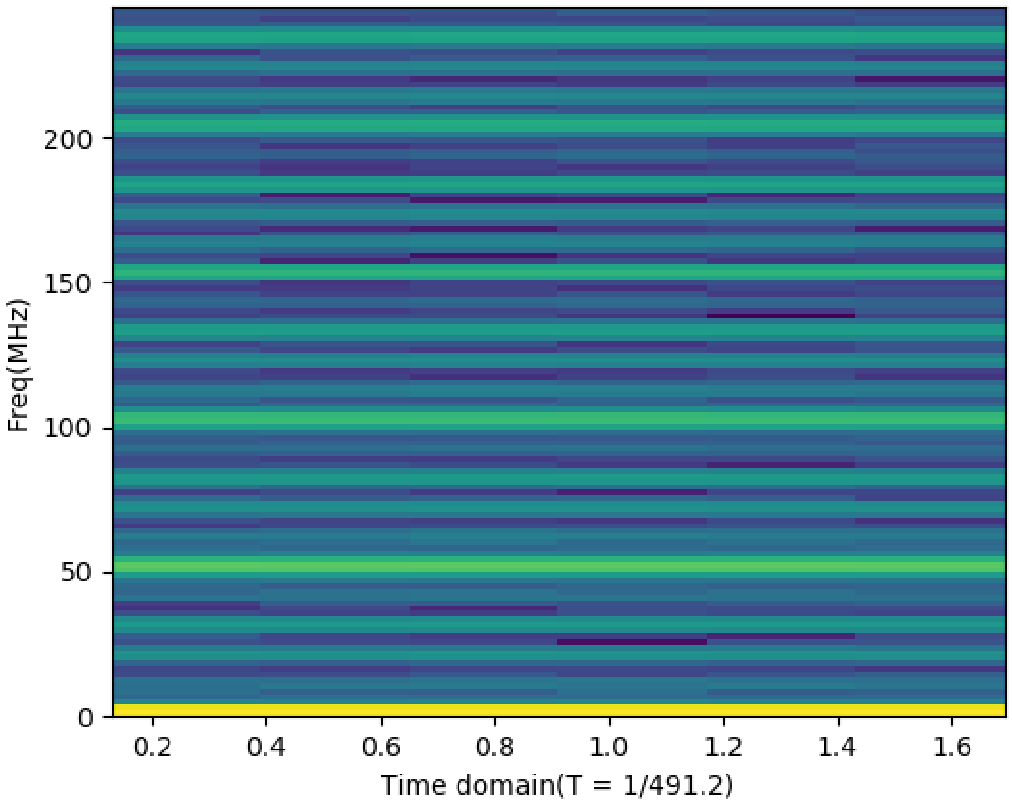
```
# plot the wave
time = np.arange(0,nframes)*(1.0 / framerate)
plt.plot(time,waveData)
plt.xlabel("Time(s)")
plt.ylabel("Amplitude")
plt.title("The wave data signal of single channel")
plt.grid('on')
plt.show()
```

Test Results

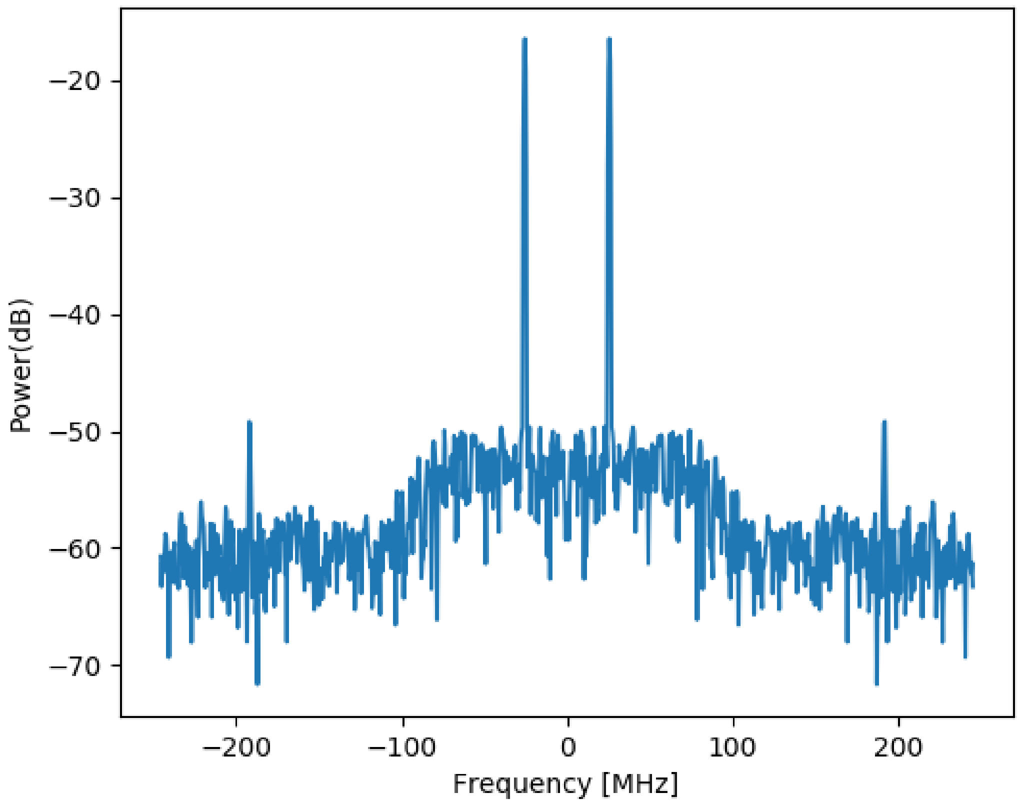
I input a ‘.wav’ file which is the record of the songs and the time length is 23 seconds. The result photo shows that the value of amplitude in each time.



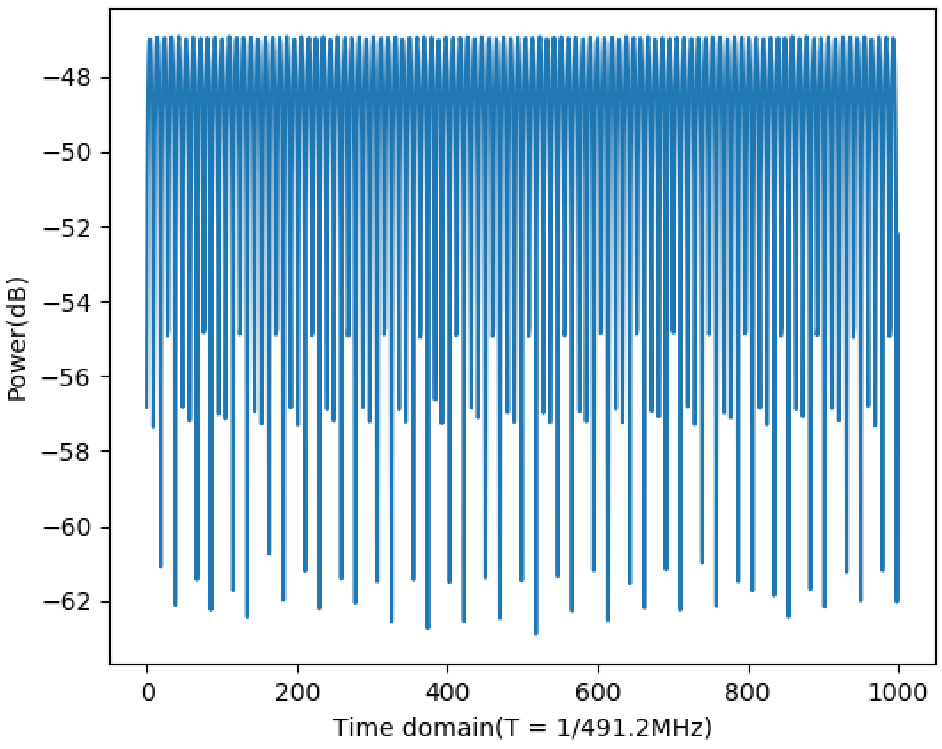
Single channel wavedata
Amplitude
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
5
10
15
20
Time(s)



Freq(MHz)
200
150
100
50
0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Time domain(T = 1/491.2)



Power(dB)
−20
−30
−40
−50
−60
−70
−200
−100
0
100
200
Frequency [MHz]



Power(dB)
−48
−50
−52
−54
−56
−58
−60
−62
0
200
400
600
800
1000
Time domain(T = 1/491.2MHz)