Uuniversité de Bourgogne

Signal Processing

Electric Guitar and Saturation

Student:
Diana AVETISYAN

November 29, 2021

Contents

1 Question 1

In this project, we study the saturation effect on electric guitars. For this purpose, we have 3 different sounds corresponding to three different levels of saturation. First of all, I have imported the essential libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
import sounddevice as sd
```

Using the sf.read command I take the information from the sound, and take the 2-D array and the frequency. The array has 2 dimensions because we have a stereo sound here, if I use the headphones I'll be able to hear the sound with two ears, with a slight difference. Further on I'll be using only one of the channels to work on, using the `array = array[:,0]` syntax.

```
short, fsp1 = sf.read('NeckCleanExt.ogg')
short = short[:,0] #I take one of the channels
Ar = len(short)

shortcrunch, fsp2 = sf.read('NeckCrunchExt.ogg')
shortcrunch = shortcrunch[:,0]

shortdirt, fsp3 = sf.read('NeckDirtyExt.ogg')
shortdirt = shortdirt[:,0]
Ard = len(shortdirt)
```

Here I am defining the time array, as the length of the `Ar` and the length of the array of the crunchy sound (`Arc`) are the same, I'll use `Ar` further on (Figure 1).

Name	Type	Size	Value
Ar	int	1	132404
Arc	int	1	132404
Ard	int	1	132724

Figure 1: Lengths of arrays

2 Question 2

Afterwards I continue with defining the time arrays in order to plot the signal.

```
timep1 = np.arange(Ar)/fsp1
timep2 = np.arange(Ar)/fsp2
timep3 = np.arange(Ard)/fsp3
```

With the help of the time arrays I will proceed with plotting the signals in the time domain to see the differences between the signals.

```
fig, axs = plt.subplots(3)
fig.suptitle('Clean, Crunchy and Dirty Sounds')
axs[0].plot(timep1,short)
axs[1].plot(timep2, shortcrunch)
axs[2].plot(timep3, shortdirt)
```

The result of the code is shown in Figure 2.

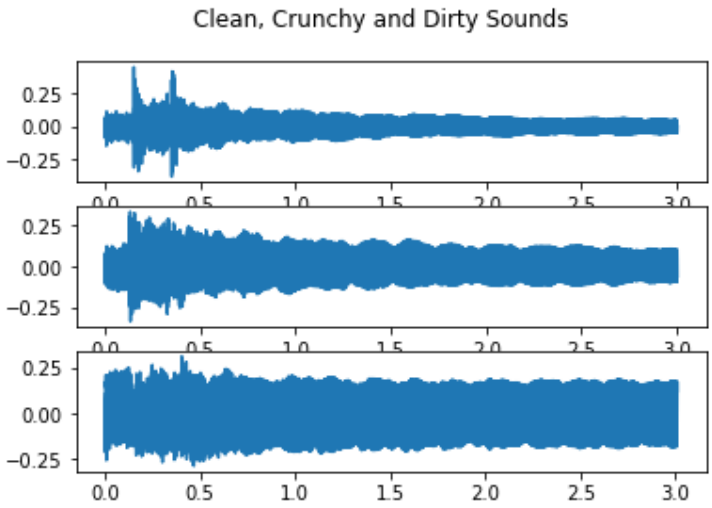


Figure 2: Clean Crunchy and Dirty Sounds

As we can see, there are additional oscillations in the crunchy and dirty sounds that are added to the sound of the electric guitar (mostly on higher notes) to make the sound deeper. This effect is called saturation.

3 Question 3

To begin with I started with loading the necessary audio files.

```
long, fsp = sf.read('NeckClean.ogg')
longcr, fspcr = sf.read('NeckCrunch.ogg')
longd, fspd = sf.read('NeckDirty.ogg')
```

Next I separated one of the channels

```
long = long[:,0]
longcr = longcr[:,0]
longd = longd[:,0]
```

Afterwards I have to define the starting point that corresponds to the 2s point ((T_s)) of the audio. As the audio length for the dirty sound differs from the clean and the crunchy sound, I will have to define it separately ((N_start) and `N_start_d`).

```
T_s = 2 #seconds (the starting point)
N_start = int(np.floor(N*T_s/53)) #the length of the audio files is 53 seconds
N_start_d = int(np.floor(N2*T_s/50)) #the length of the audio file is 50 seconds
```

Next I have splitted the array to take the interesting part which corresponds to the `[N_start:  N_start + Np]`.

```
longwincl = long[N_start:N_start+Np]
longwincr = longcr[N_start:N_start+Np]
longwind = longd[N_start_d:N_start_d+Np]
Nd = len(longwind) #Because the audio length of the dirty sound is different
```

After getting the part I need, I will perform the Fourier transform.

```
long1 = np.fft.fft(longwincl)
long2 = np.fft.fft(longwincr)
long3 = np.fft.fft(longwind)
```

4 Question 4

Using the results from the previous section, I will define the time period that will be used later to define the frequency arrays and plot the Fourier transform.

```
ts = 1/fsp
tscr = 1/fspcr
tsd = 1/fspd

freqsp = np.arange(Np)/(ts*Np)
freqspc = np.arange(Np)/(tscr*Np)
freqspd = np.arange(Np)/(tsd*Np)
```

And the plotting part

```
n = 5500 #to zoom in I used n=5500 or 500
long1 = long1[0:n]
long2 = long2[0:n]
long3 = long3[0:n]

freqsp = freqsp[0:n]
freqspc = freqspc[0:n]
freqspd = freqspd[0:n]

plt.plot(freqspd, np.log10(np.abs(long3)))
plt.plot(freqspc, np.log10(np.abs(long2)))
plt.plot(freqspc, np.log10(np.abs(long1)))
```

There are some more or less visible well-defined frequencies in the $200 - 800Hz$ range. They are more visible for the clear sound, and it becomes slightly less visible in the dirty and crunchy sounds (Figure 4).

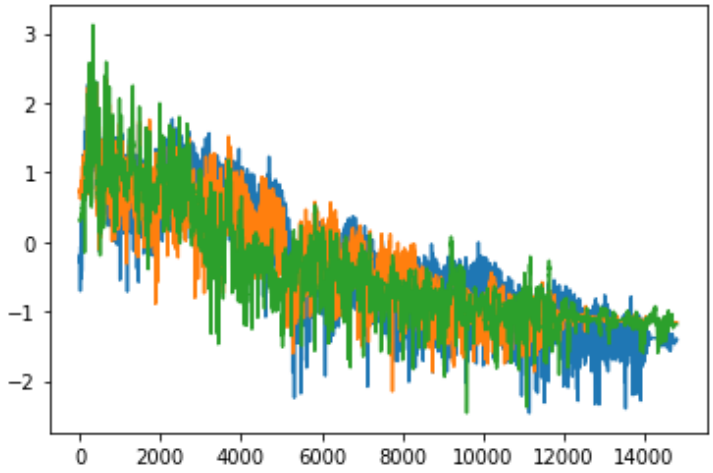


Figure 3: Plot of the Fourier transform of the sounds. The green plot corresponds to the clean sound, the orange is the crunchy and the blue is the dirty sound

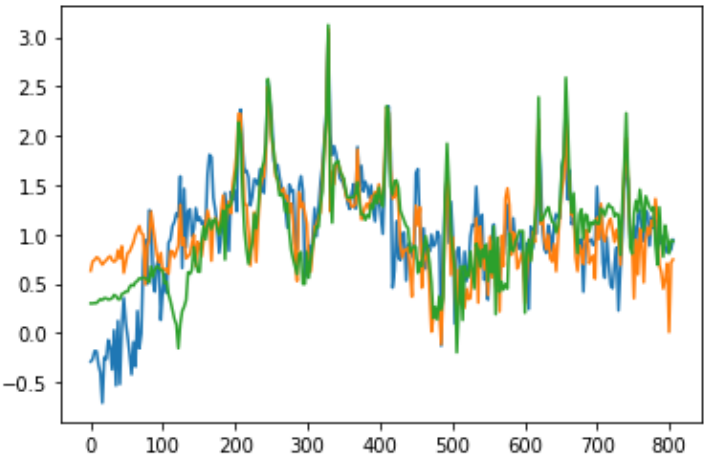


Figure 4: Zoomed in plot

5 Question 5

In order to model what the saturation curve is, in essence I will define an array of 2^{14} zeros, and perform a $\sin(\omega t)$ function on it afterwards. I'll take $\omega = 440Hz$.

```
Ns = 2**14
time = np.arange(Ns)
x = np.zeros(Ns)
for i in time:
    x[i] = np.sin(440*i)
#Performing a Fourier transform on x(t)
xf = np.fft.fft(x)
```

After performing the Fourier transform on $x(t)$, the problem is to plot it as the function oscillates very fast, that's why I used some tricks to cut the array and plot only a small part of it.

```
l = len(t)
t = np.linspace(0, (6*np.pi)/440)
xc = x[0:l]
fig1, axs1 = plt.subplots(2)
fig1.suptitle('The sine signal and its Fourier transform')
axs1[0].plot(t,xc)
axs1[1].plot(xf)
```

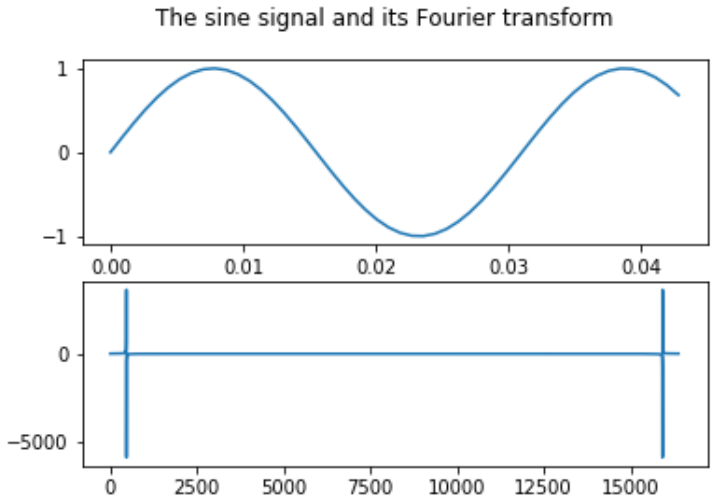


Figure 5: The plot of the $\sin(440t)$ and its Fourier transform

As we see the Fourier transform of the sine function is a sum of two delta functions. which I have also calculated analytically (equations (1)-(3)) As

$\sin(\omega_0 t)$ is not an absolutely integrable function and hence I cannot use the formula of Fourier transform.

$$\sin(\omega_0 t) = \frac{1}{2i}(e^{i\omega_0 t} - e^{-i\omega_0 t}) \tag{1}$$

$$FT\sin(\omega_0 t) = \frac{1}{2i}(FT(e^{i\omega_0 t}) - FT(e^{-i\omega_0 t})) \tag{2}$$

$$x(\omega) = frac12i(\delta(\omega - \omega_0) - \delta(\omega + \omega_0)) \tag{3}$$

So the plotted result is the same as the result that I calculated analytically.

6 Question 6

To begin with, I defined the following function where α is a positive constant

$$y(t) = \begin{cases} x(t) & \text{if } |x(t)| \leq \alpha \\ \alpha & \text{if } |x(t)| > \alpha \end{cases}$$

smaller then 1.

```
alfa = 0.5
def saturation(a):
        return a if np.abs(a)<= alfa else alfa
```

Now the defined function should be performed on a 1D array of `Ns` elements.

```
y = np.zeros(Ns)
for i in time:
    y[i] = saturation(x[i])
yc = y[0:1]
plt.plot(t,xc)
plt.plot(t,yc)
```

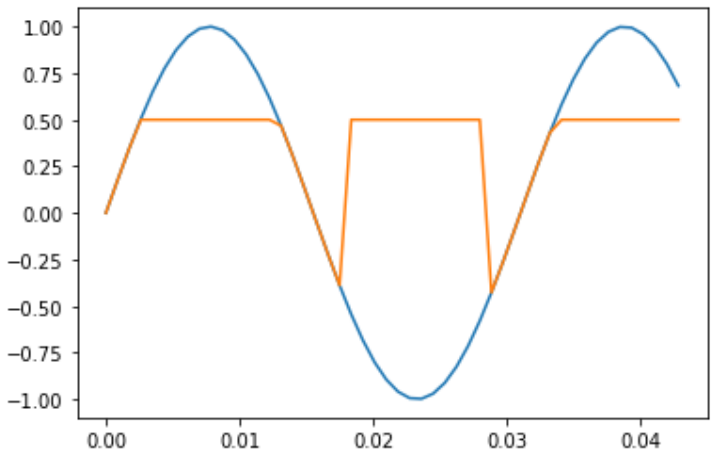


Figure 6: Plot of the original and saturated signals

Also I like to play the sound to feel the difference, attention, ear piercing sounds :)

```
ext = np.concatenate((x, y))
sd.play(ext)
```

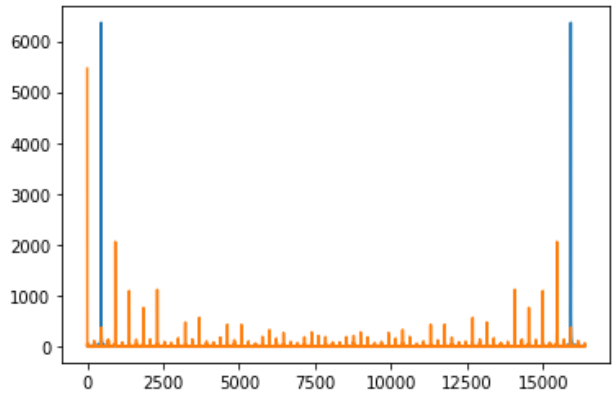
The definition of saturation of the sound is adding additional frequencies to the existing clear sound. Here I have cut out a specific part of the function, and replaced it with a constant value instead. Constant signal is periodic as it has an uncountable infinite number of periods, but it does not have a fundamental period as such. But the good news is that the effect of saturation on this signal can clearly be observed after playing the sound. As a result of this transformation the sound becomes richer.

7 Question 7

Please find below the code for finding the Fourier transform of `x` and `y`.

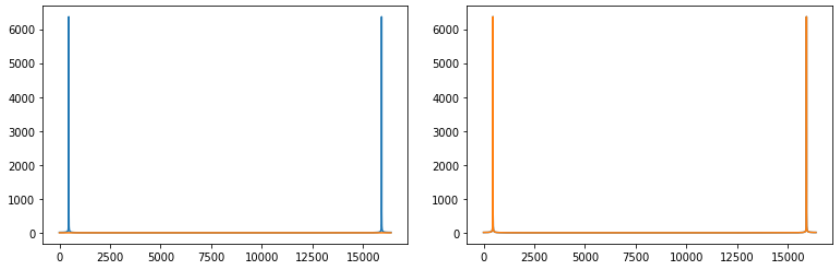
```
yf = np.fft.fft(y)
xf = np.fft.fft(x)
plt.plot(np.abs(xf))
plt.plot(np.abs(yf))
```

Figure 7: The Fourier transform of the original sound and its saturated version



In the Figure 7 `alfa = 0.5` and as a result we can see additional peaks despite the two main delta functions that correspond to the Fourier transform of the sine function. If the `alfa` goes to zero, the intensity decays, and as a result we will get a mute sound when `alfa` $= 0$ (Figure 8). When `alfa` goes to 1, the additional frequencies and consequently the saturation go to zero, so as a result the sound becomes clearer (Figure 8). Saturation can not be achieved by using any type of filter, because the saturation is adding new frequencies to the existing signal and the filter on the contrary, removes the unnecessary frequencies.

Figure 8: The behaviour of the curve depending on `alfa`



(a) `alfa = 0.00001` (b) `alfa = 0.999`

References

https://github.com/UndineStein/Electric-guitar-and-saturation