Assignment 1, Digital Signal Processing: Fourier Transform University of Glasgow School of Engineering

Katarzyna Lenard & Mustafa Biyikli

Introduction

Vocals can be enhanced by identifying the fundamental and harmonic frequency ranges and manipulating certain frequencies. This study describes a simple harmonic enhancer script and its abilities written in Python programming language resembling the frequency response of Shure SM-58 vocal microphone.

The aim of this study was to improve the sound quality of a 23-year-old male subject, by enhancing its harmonics. The study was constrained to post-editing of a pre-recorded WAV file at a minimum of 44kHz sampling rate and only using simple Python commands: Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) from the numpy library.

The objectives were to plot the audio signal both in the time and frequency domains, identify its fundamental and harmonic frequency ranges and to manipulate the frequency response to resemble the signal characteristics of a professional vocal microphone as shown in Figure 1.

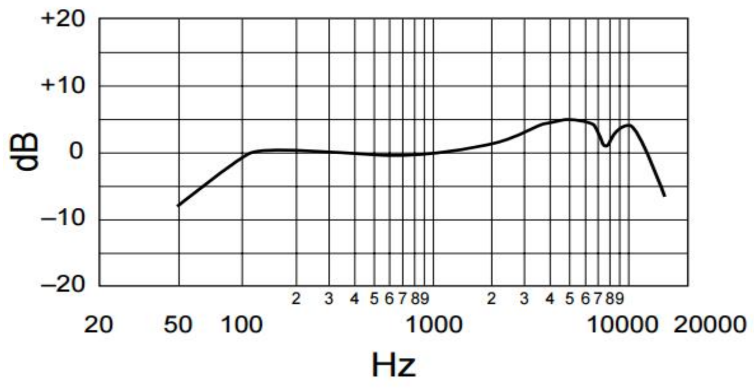

Figure 1, Shure SM-58 frequency response

Methods

Data Acquisition & Presentation

The voice of a male subject (age 23) was recorded while singing at a sampling rate of 48kHz using a mobile phone microphone and imported into Python as a WAV file. The recording consisted of 2 channels of datatype 16-bit integer consisting of 2^{16} (65536) quantisation levels, corresponding to theoretical minimum and maximum values of $\pm 2^{15}$. For the purposes of this study, channel-2 data was removed, and channel-1 data was normalised by diving the signal amplitude by the maximum theoretical value.

The normalised data for the raw audio signal was plotted against time in seconds which was calculated using equation 1 where T_{max} is the total recording time, N is the total number of samples and Fs is the sampling frequency.

$$T_{max} = \frac{N}{Fs} \tag{1}$$

The audio signal was transformed from the time domain into the frequency domain using the FFT command. Following the definition of the discrete Fourier Transform, to plot the frequency domain, the signal was divided by the number of samples, making it independent of the sample size, as this is not performed by the FFT. This is shown in equations 2 and 3, where N stands for the total number of samples, n stands for the sample index, and k is the frequency index.

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \tag{2}$$

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k)e^{j2\pi kn/N} \tag{3}$$

It can be seen from equations 2 and 3 that this process creates an array of real and imaginary numbers, which cannot be plotted on real axes. Therefore, for the purposes of plotting, the imaginary part of the array was disregarded.

Prior to plotting, the data was converted into Decibels (dB) using equation 4 and a graph of amplitude [dB] against frequency [Hz] was plotted on a logarithmic frequency axis.

$$x(f)_{dB} = 20log_{10}(x(f)) \tag{4}$$

The frequency, x-axis was limited to the Nyquist frequency, shown in equation 5. This was performed to exclude the mirrored part of the frequency spectrum.

$$f_{max} = \frac{Fs}{2} \tag{5}$$

Identifying the Fundamental and Harmonic Frequencies

Fundamental frequencies were identified, using the frequency domain plot, to be in the range of 100Hz to 1kHz as there were significant peaks visualised within this region. Within this range, several different fundamental frequencies were found caused by the different words sang throughout the recording.

The harmonics were identified to be from 1kHz to 10kHz visualised as smaller peaks corresponding to the previously identified fundamental frequencies. These values were justified by comparison to those found in the existing literature as shown in Table 1.

Table 1, Typical frequency ranges found in literature compared to the findings of this study

Frequency Ranges [Hz]	This experiment	Kreiman and Gerratt (2012)	Jeng et al. (2011)
Fundamental Frequency Range	100Hz - 1kHz	126Hz - 545Hz	170Hz - 680Hz
Harmonic Frequency Range	1kHz - 10kHz	1.2kHz - 4.3kHz	1kHz - 1.4kHz

Harmonic Enhancer Frequency Response

The frequency response range of the enhancer was limited to 50Hz – 15kHz based on the frequency response of Shure SM-58. Therefore, a high pass filter was applied to remove any frequencies below 50Hz. Similarly, a low pass filter was applied to remove any frequencies above 15kHz.

Specific ranges of the frequency spectrum were enhanced individually. Therefore, several loops were created to resemble the frequency response of the Shure SM-58 shown in Figure 1. The operations were following a linear interpolation. First, in the range of 50Hz to 100Hz there was a gradual removal of 8dB to none. 100Hz to 1kHz region remained untouched as they were identified to be the fundamental frequencies. Frequencies above 1kHz were boosted gradually from none to 5dB at 5.5kHz. The same 5dB boost was kept up to 6kHz. These high frequencies were full of harmonics and needed enhancing. From 6kHz to 8kHz the signal was boosted in a decreasing pattern from 5dB to 1dB and boosted again in an increasing pattern to 4dB at 10kHz. At 10kHz a sharp frequency drop was identified in the spectrum indicating the end of the vocal frequency range. Therefore, the signal amplitude was shifted from boosting 4dB to cutting 7dB at 15kHz, which corresponded to the upper limit of the enhancer frequency response. The operations performed are plotted on an enhancer frequency response graph shown in Figure 2.

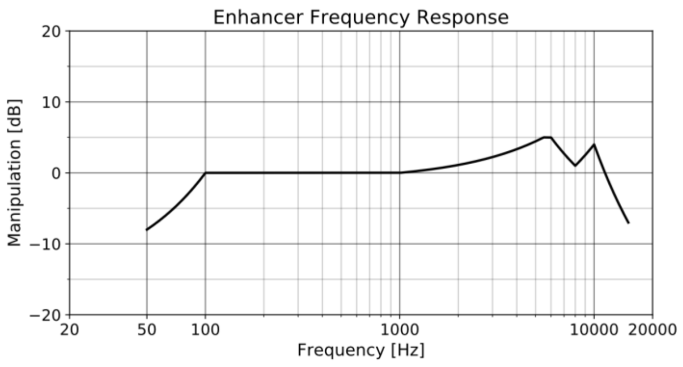

Figure 2, Enhancer frequency response

Both high and low pass filters as well as the boosting and cutting operations were applied to both complex conjugates of the frequency spectrum, as shown in equation 6.

$$x^*(k) = x(-k) = x(N-k) \tag{6}$$

This method was followed to ensure that the output audio was only real valued cancelling out the imaginary parts of the signal. The improved frequency domain was plotted for comparison to the original frequency domain.

Saving an Improved Audio WAV File

Following the enhancement process, the improved signal was transformed back to the time domain using the IFFT command. The improved signal was plotted in the time domain and compared to the original signal. The data was written to a new WAV file with datatype 16-bit integer. Original and improved audio files were listened and compared.

Results

Four plots were produces as described previously. Figure 3 (a) shows the original signal in the time domain and Figure 3 (b) corresponds to its frequency domain.

Figure 3 (c) and (d) are the improved signal plots in the frequency and time domains, respectively. In the frequency domain plots for both original and improved signals, a dashed line at -75dB was plotted for ease of interpretation and comparison. As it can be seen by comparing Figure 3 (b) and (c) the results follow the harmonic enhancer frequency response presented in Figure 2.

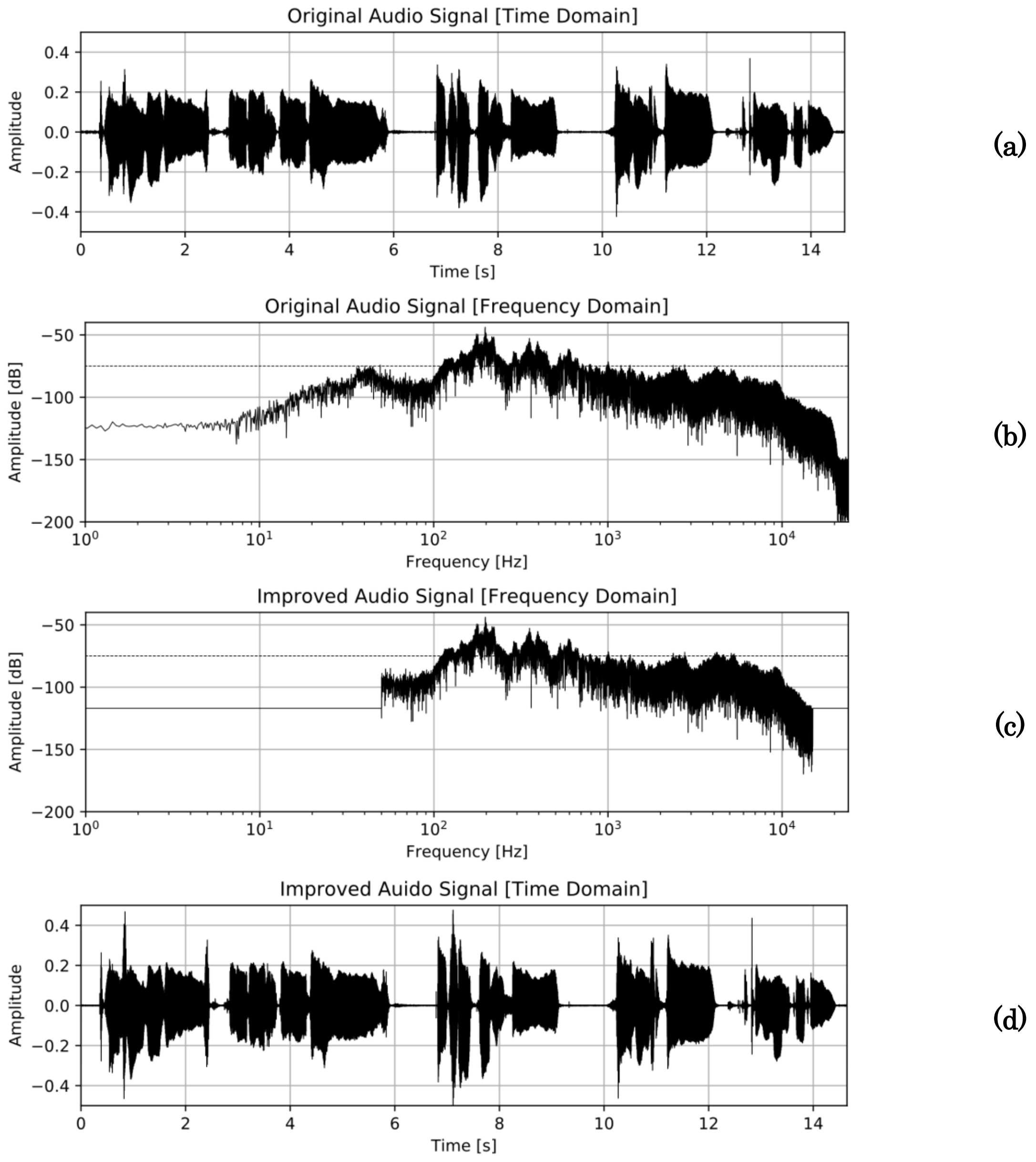

Figure 3, Original and improved signals in time and frequency domains

Additionally, spectrograms for both signals were plotted to validate the results for original and improved signals. These are shown in Figure 4 (a) and (b), respectively. Complete code used to achieve these results can be found in Appendix A.

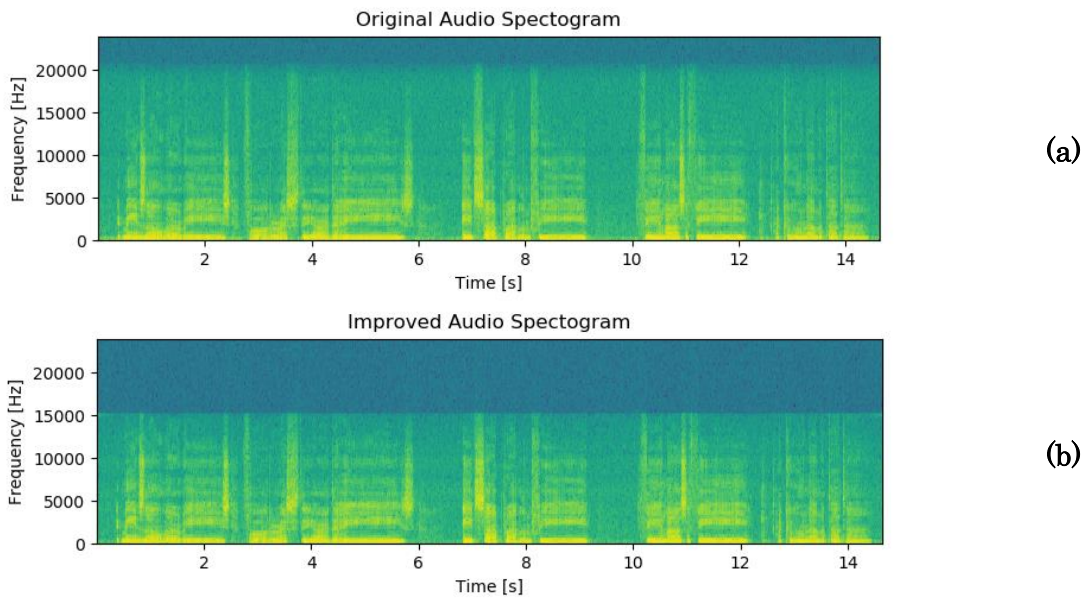

Figure 4, Original and improved audio spectrogram

Discussion

As it can be seen in Figure 3 (b), the original audio signal and (c), the improved audio signal, the harmonic frequency ranges from 1kHz to 10kHz were boosted following the enhancer frequency response and the background noises were filtered out. The improved audio crosses the reference line placed at -75dB where the original audio signal falls below the -75dB line for the harmonic peaks. Having listened the audio files, clear differences were spotted in the voice quality. In comparison, the original audio was dull, and the improved audio signal produced a sharper and clearer tone. In Figure 3 (d), the harmonics can be identified in the time domain by comparing the signal amplitude to Figure 3 (a). As the harmonic frequencies were amplified, the improved signal possesses higher amplitudes at these ranges.

As the recording sampling frequency, Fs, was very high, there were no ambiguities, which can be seen in the audio spectrograms presented in Figure 4 (a). All vocal frequencies fell below the Nyquist frequency, Fs/2, which suggested that 48kHz is a sufficient sampling rate for recording male vocals.

Although not identical, the frequency response of the harmonic enhancer was very similar to the frequency response of Shure's SM-58 vocal microphone. This can be visualised by comparing Figure 1 and Figure 2. With further refinement of the enhancer frequency response, the vocals can be improved to sound better, however, this method cannot be applied to any real-time applications as the Fourier Transform is a non-causal method of signal processing. Therefore, it requires all the samples to produce the spectrum as it can be seen from equations 2 and 3 mentioned previously.

Conclusions

In this study, a Python based harmonic enhancer was developed using simple commands and tested with the vocals of a 23-year-old male recorded at 48kHz sampling rate. The fundamental and harmonic frequencies were identified and enhanced following the frequency response of a professional vocal microphone, Shure SM-58. The enhancer script had a frequency response range of 50Hz to 15kHz and it kept the fundamental frequencies untouched and enhanced the harmonics in a specified ramp pattern. Although not identical to the frequency response of SM-58, a very similar enhancer frequency response was achieved. This was done by transforming the signal from time to frequency domain using Fast Fourier Transform, modifying the frequency spectrum and transforming it back to the time domain by using the Inverse Fast Fourier Transform. Improvements made on the original audio were clearly visible on the plots and audible to human ear. Improved audio was sharper and clearer yet retained its warmth.

It was discussed that this method of audio enhancement can only be applied for a pre-recorded signal as the Fourier Transform is non-causal. Therefore, the present response of this enhancer depends on the future values of the inputs. Most applications requiring real time processing of the signal would not be able to use the enhancer presented in this study. An improvement on this study would be to implement a similar frequency response using causal methods of signal processing, such as the finite impulse response (FIR) filters.

References

Jeng, F.-C., Costilow, C. E., Stangherlin, D. P. & Lin, C.-D. 2011. Relative Power of Harmonics in Human Frequency-Following Responses Associated with Voice Pitch in American and Chinese Adults. *Perceptual and Motor Skills,* 113, 67-86.

Kreiman, J. & Gerratt, B. R. 2012. Perceptual interaction of the harmonic source and noise in voice. *Journal of the Acoustical Society of America,* 131, 492-500.

Appendix A

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 10 15:12:54 2019

@authors: Katarzyna Lenard - 2218524L & Mustafa Biyikli - 2190523B
"""

import scipy.io.wavfile as wavfile
import numpy as np
import matplotlib.pyplot as plt

Fs, x_t = wavfile.read("original.wav")      # Load the audio sample
x_t = x_t[:, 1] / 2**15                     # Reduce to single-channel and
normalise the signal (16-bit recording)
x_t = x_t[50000 : len(x_t)-35000]           # Remove muted sections
[beginning & end]
resolution = Fs/len(x_t)
margin = -75                                # Margin line on
amplitude[dB]/frequency[Hz] graph
y = np.linspace(margin, margin, len(x_t))   # This is to plot the margin
line later on
figManager = plt.get_current_fig_manager()
figManager.resize(*figmanager.window.maxsize())             # Load the plots
in a maximised window (changed for Linux OS)
SM58_mimic = []                             # This is used to plot the
enhancer frequency response by appending ramp values to it

# Plot the audio signal against time, add title and name x & y axes
time = np.linspace(0, len(x_t)/Fs, num=len(x_t))    # Create an array of
time[s] using number of samples
plt.subplot(321, facecolor="white") and plt.plot(time, x_t, "black",
linewidth=0.5)
plt.title("Original Audio Signal [Time Domain]", loc="center")
plt.ylabel("Amplitude") and plt.xlabel("Time [s]")
plt.xlim(0, len(x_t)/Fs) and plt.ylim(-0.5, 0.5)
plt.grid()

# Plot the audio signal against frequency, add title and name x & y axes
x_f = np.fft.fft(x_t)  # Forward FT: Time Domain -> Frequency Domain; Make
data sample size independent
x_f_plot = 20*np.log10(np.fft.fft(x_t)/len(x_t))    # Discrete FT for the
plot
x_f_dB = 20*np.log10(x_f)   # Convert to dB
frequency = np.linspace(0, Fs, len(x_t))    # Create an array of
frequency[Hz] using Nyquist Theorem of Fmax <= Fs/2
plt.subplot(322, facecolor="white") and plt.plot(frequency,
np.real(x_f_plot), "black", linewidth=0.5)
plt.xscale("log")
plt.title("Original Audio Signal [Frequency Domain]", loc="center")
plt.ylabel("Amplitude [dB]") and plt.xlabel("Frequency [Hz]")
plt.ylim(-200, -40)
plt.xlim(1, Fs/2) # Exclude the mirrored part from the log scale start is 1
as 10**0 = 1
plt.grid()
plt.plot(frequency, y, "k--", linewidth=0.5)    # This is the margin line

#Apply a High Pass Filter (50Hz)
h1 = int(len(x_f_dB)/Fs*0)
h2 = int(len(x_f_dB)/Fs*50)
x_f_dB[h1 : h2+1] = 0
x_f_dB[len(x_f_dB)-h2 : len(x_f_dB)-h1+1] = 0

# Apply a Low Pass Filter (15kHz)
l1 = int(len(x_f_dB)/Fs*1.5e4)
l2 = int(len(x_f_dB)/Fs*Fs/2)
x_f_dB[l1 : l2+1] = 0
x_f_dB[len(x_f_dB)-l2 : len(x_f_dB)-l1+1] = 0

# 50Hz to 100Hz Region, SHURE SM-58 frequency response manipulation (Remove
+8dB from 50Hz and 0dB from 100: linear correlation)
for harmonics in np.arange(50, 1e2, resolution):
    ramp = harmonics*8/50 - 16
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    SM58_mimic.append(ramp)

# Leave 100Hz to 1kHz region untouched
for harmonics in np.arange(1e2, 1e3, resolution):
    ramp = 0
    # Uncomment the following if fundemetal region is to be modified;
change ramp value as desired
    '''
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    '''
    SM58_mimic.append(ramp)

# 1kHz to 5.5kHz Region, SHURE SM-58 frequency response manipulation (Add
0dB to 1kHz and +5dB to 5.5kHz: linear correlation)
for harmonics in np.arange(1e3, 5.5e3, resolution):
    ramp = harmonics/900 - 10/9
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    SM58_mimic.append(ramp)
```

```
# 5.5kHz to 6kHz Region, SHURE SM-58 frequency response manipulation (Add
+5dB from 5kHz to 6kHz: linear correlation)
for harmonics in np.arange(5.5e3, 6e3, resolution):
    ramp = 5
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    SM58_mimic.append(ramp)

# 6kHz to 8kHz Region, SHURE SM-58 frequency response manipulation (Add
+5dB to 6kHz and +1dB to 8kHz: linear correlation)
for harmonics in np.arange(6e3, 8e3, resolution):
    ramp = 2.4e4*4/harmonics - 11
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    SM58_mimic.append(ramp)

# 8kHz to 10kHz Region, SHURE SM-58 frequency response manipulation (Add
+1dB to 8kHz and +4dB to 10kHz: linear correlation)
for harmonics in np.arange(8e3, 1e4, resolution):
    ramp = harmonics*3/2000 - 11
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    SM58_mimic.append(ramp)

# 10kHz to 15kHz Region, SHURE SM-58 frequency response manipulation (Add
+4dB to 10kHz and remove +7dB from 15kHz: linear correlation)
for harmonics in np.arange(1e4, 1.5e4, resolution):
    ramp = 1.5e4*11/harmonics*2 - 29
    k1 = int(len(x_f)/Fs*(harmonics))
    x_f_dB[k1] = x_f_dB[k1] + ramp
    x_f_dB[len(x_f_dB)-k1] = x_f_dB[len(x_f_dB)-k1] + ramp
    SM58_mimic.append(ramp)

# Plot the improved audio signal against frequency, add title and name x &
y axes
x_f_plot = 10**(x_f_dB/20)
plt.subplot(324, facecolor="white") and plt.plot(frequency,
np.real(20*np.log10(x_f_plot/len(x_t))), "black", linewidth=0.5)
plt.xscale("log")
plt.title("Improved Audio Signal [Frequency Domain]", loc="center")
plt.ylabel("Amplitude [dB]") and plt.xlabel("Frequency [Hz]")
plt.ylim(-200, -40)
plt.xlim(1, Fs/2) # Remove the mirrored part from the log scale start is 1
as 10**0 = 1
plt.grid()
plt.plot(frequency, y, "--k", linewidth=0.5)
```

```
# Transform the improved audio back to time domain
sound_clean = np.fft.ifft((10**(x_f_dB/20))*(2**15)) # Convert back to x[t]
sound_clean = np.real(sound_clean)
sound_clean = np.asarray(sound_clean, dtype=np.int16)

# Plot the improved audio signal against time, add title and name x & y
axes
plt.subplot(323, facecolor="white") and plt.plot(time,
sound_clean/2**15, "black", linewidth=0.5)
plt.title("Improved Auido Signal [Time Domain]")
plt.ylabel("Amplitude") and plt.xlabel("Time [s]")
plt.xlim(0, len(x_t)/Fs) and plt.ylim(-0.5, 0.5)
plt.grid()

# Not necessary, plotted just to justify
plt.subplot(325)
plt.specgram(x_t, Fs=Fs)
plt.title("Original Audio Spectrogram")
plt.ylabel("Frequency [Hz]") and plt.xlabel("Time [s]")

# Plot the enhancer frequency response which shows how the signal is
manipulated
plt.subplot(326, facecolor="white") and plt.plot(np.linspace(50, 15000,
num=len(SM58_mimic)), SM58_mimic, "black", linewidth=2)
plt.xscale("log")
plt.title("Enhancer Frequency Response")
plt.ylabel("Manipulation [dB]") and plt.xlabel("Frequency [Hz]")
plt.xlim(10**0, Fs/2)
xmajor_ticks = [20, 50, 100, 1000, 10000]
ymajor_ticks = np.arange(-10, 11, 5)
plt.xticks(xmajor_ticks, xmajor_ticks)
plt.yticks(ymajor_ticks, ymajor_ticks)
plt.tick_params(axis='both', which='major')
plt.grid(which='both', alpha=1)

# Titles and axes labels overlap, this solves them; could also use
plt.tight_layout()
plt.subplots_adjust(wspace=0.25, hspace=0.6)
plt.show()

wavfile.write("improved.wav", Fs, sound_clean)  # Save the improved audio
in the same directory
```