Lecture 2: Digital Filters in Audio

Introduction

The lecture covered various digital filters which are commonly used tools in audio signal processing. In particular, it gave an overview FIR and IIR filters, comb filters, resonators, allpass filters, shelving filters, equalizing filters and fractional delay filters. Moreover, filter implementations and characteristics were accompanied by examples and a variety of use cases. Through this diary I will try to provide a brief overview of each of the filter types covered in the lecture and their use cases. At the end of the diary a Python implementation of an FIR low-pass filter is also included.

Overview

FIR and IIR Filters

FIR (Finite Impulse Response) filter is a type of filter whose impulse response is finite, meaning that when the filter is subjected to a unit impulse after some time its output will eventually settle to zero.
On the other hand, there also exists another type of filters known as **IIR (Infinite Impulse Response)** filters. Contrary to FIR filters, their impulse response is infinite due to the recursive nature of their design. This means that although their output might eventually become infinitesimally small, it actually never reaches exact zero like that of an FIR filter.

Comb Filters

Comb filter is a filter in which we add a delayed version of the signal to itself. This results in a comb-like shapes both of the magnitude and frequency responses of the filter. Moreover, by changing the sign of the feedback coefficient a_1 we can control at which harmonics the peaks occur (odd harmonics for negative sign and even harmonics for the positive sign). Besides that, it is also possible to create an inverse comb filter by inverting the comb filter transfer function. This results in an inverted comb shape of the filter response. These filters have various applications ranging from delay to flanger effects, acoustic standing wave modelling, etc.

Resonators

Digital resonator is most commonly designed using a two pole approach where first pole is set and the second pole is its complex conjugate. While this approach implies that we can just simply select θ and $R = 1 - \frac{B}{2}$ parameters, this can cause an issue at low frequencies as the pole and it's complex conjugate are close to each other and can cause bias at the peak location. To alleviate this issue Steiglitz proposed a method that uses correction term - $cos(\theta) = [(1 + R^2)/2R]cos(\psi)$ where ψ is the peak frequency.
Besides digital resonator filters, these types of filters can also be encountered in nature, particularly in the way how we synthesize speech. We can view speech synthesis as a source-filter model where our vocal folds serve as source while our vocal tract acts as a filer to articulate our speech. Here, the column of air in the vocal tract acts as a resonator filter whose parameters are modified by the shape of the vocal tract.

Allpass Filters

Allpass filter, as its name suggests, is a filter which does not attenuate nor boost anything resulting in its magnitude response to be 1 at all frequencies. Although at first we might question why this filter even exists, there's a range of various use cases where it can be applied. For instance, we can use it for phase correction, reverberation and delay, phaser effect by combining original signal with slightly lagged copy of itself, etc.

Shelving Filters

Shelving filter is a type of filter that boosts or attenuates low or high frequencies without affecting their counterparts, can be viewed as a simple bass or treble control. Low shelving filter can be especially useful in music production to eliminate unwanted rumbling and inaudible frequencies in the low end of spectrum since they can often make recordings clip whilst not bringing any improvement to the overall listening experience.

Equalizing Filters

Equalizing filter is probably the most known filter due to its wide use in modern audio production. It is commonly used to change magnitude response of an audio track, often to make the spectrum bumpy so certain frequency ranges stand out more than others, e.g. bass boosting where magnitude of low frequencies is increased. However, contrary to today's common use cases, its initial purpose was not to make the spectrum bumpy, but rather flat, which was used for flattening frequency response of signals in telephone lines.

Fractional Delay Filters

Fractional delay filters are specific due to the fact that the delay within them is smaller than the sample interval. Consequently, in the case when dealing with a discrete signal we cannot simply access these values as they're not available in the sampled signal. In order to circumvent this issue we use various methods to estimate signal value at that time point. These methods include linear interpolation, cubic interpolation or higher order polynomial fitting which we then use to extract the desired value at a given time point.

FIR Low-Pass Filter Implementation Example

In this example I implemented an FIR low-pass filter using window design approach with a Kaiser windowing method. We set a cutoff frequency at 1000 Hz and a transition width at 100 Hz where stop frequencies, i.e. those above 1000 Hz are attenuated by 60 dB. In Figure 1 we can see that the original signal is much richer in high frequency content when compared to the filtered signal. For the input of this example I used the famous jazz phrase, commonly referred to as "The Lick"[1].

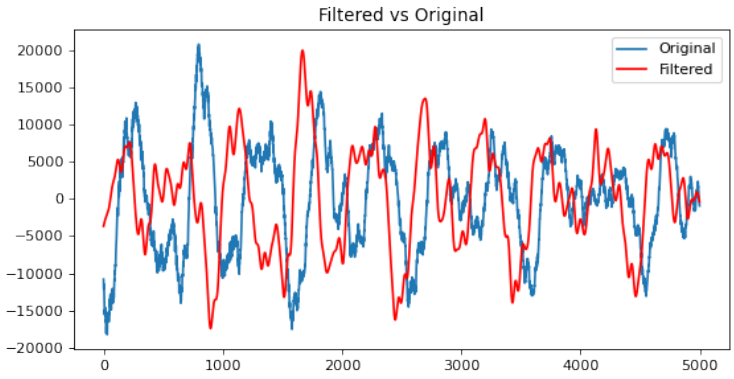

Figure 1: Comparison of original and filtered signals at an arbitrary range in the signal (240000 - 245000 sample).

```python
import numpy as np
from scipy import signal
from scipy.io import wavfile
import IPython.display as ipd
import matplotlib.pyplot as plt

def lowpass_fir(sample_rate, x):
    nyq_freq = sample_rate * 0.5
    transition_width = 1e2 / nyq_freq

    stop_attenuate_db = 60.0
    N, beta = signal.kaiserord(stop_attenuate_db, transition_width)
    cutoff_freq = 1e3
    taps = signal.firwin(N, cutoff_freq/nyq_freq, window=("kaiser", beta))
    return signal.lfilter(taps, 1.0, x)

# load data
sample_rate, data = wavfile.read("lick.wav")

# convert to mono
data = data.sum(axis=1) / 2
filtered = lowpass_fir(sample_rate, data)
ipd.Audio(filtered, rate=sample_rate)
```

```python
plt.figure(figsize=(8, 4), dpi=80)
plt.plot(data[240000:245000])
plt.plot(filtered[240000:245000], "r-")
plt.legend(["Original", "Filtered"])
plt.title("Filtered vs Original")
plt.savefig("comparison.png")
plt.show()
```

References

[1] Wikipedia - The Lick