FDST LAB: Assignment #2
One-pole lowpass filter

Due Date:

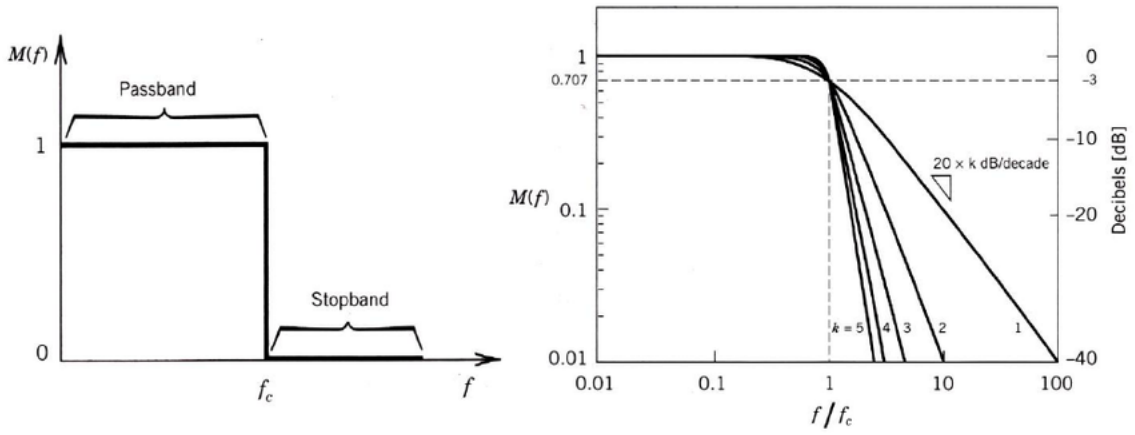
Mar. 25, 2024 @11:59PM

Instructions:

- Upload a .zip file containing your completed **python notebook file**, the given audio files **'sawtooth.wav'** and **'whitenoise.wav'**, as well as **your audio file from question 2** to Brightspace

Filtering is the process of attenuating or accentuating certain frequencies of an input signal. A common filter encountered in the world of audio signal processing is the low-pass filter. A low-pass filter allopws frequencies below a certain *cutoff frequency* to pass and removes frequencies above this frequency. An ideal filter will perfectly pass any frequencies below the cutoff and remove frequencies above it, but real-world filters are not able to do this and instead smoothly transition from the *passband* (region below the cutoff frequency) and the *stopband* (region above the cutoff frequency). The more complex the filter you build, the more closely you can approximate the ideal low-pass filter's response.

Low-pass filters are frequently used in equalizers and as anti-aliasing filters to prevent high frequency distortion that results from our limited sample rate and the Nyquist-Shannon sampling theorem.



In this assignment, you will create a function that implements the most basic low-pass filter; the one-pole lowpass filter. The filter is described by the following difference equation:

$$y[n] = d * x[n] + (1 - d)y[n - 1]$$

y[n] is the output of the filter
x[n] is the input of the filter
d is the decay factor of the filter, a value between 0 and 1

Note that at each step, the output of the filter depends on both the input signal and the previous step's output, meaning that this filter has a feedback component.

The variable *d* is chosen by the designer (you) when implementing this filter and has a large effect on the character of the filte. When d is one, your output signal perfectly matches the input signal but as d is decreased towards zero, the filter begins taking a weighted average between the input signal and the output of the previous step of the filter, smoothing out any quick changes in the input signal and effectively attenuating high frequency content.

1. Filtering waveforms (40pts)

- Read and normalize 'sawtooth.wav' (5pts)
- Code a function that applies a one-pole low-pass filter to your signal with a response determined by an argument called *decay*. (30 pts)
- Run the test code to visually and aurally inspect your low pass filter.
- Answer questions in notebook

2. Filtering complex audio (10pts)

- Apply your filter to any short mono audio file of your own. Not the provided ones (IMPORTANT: Make sure you include this with your assignment submission)
- Discuss the effects of different decay values on the sound in the notebook (10pts)

3. Implementing cutoff frequency (50pts)

The equation below is a way to calculate an approximate value for *d* from a desired cutoff frequency in Hz.

$$d = \frac{2\pi\Delta_T f_c}{2\pi\Delta_T f_c + 1}$$

where : d = decay factor
ΔT = sampling period (Not sampling rate!)

fc = cutoff frequency

- Given the above equation relating decay to sample rate and cutoff frequency, modify your function to take input arguments of a desired cutoff frequency as well as the sample rate of your input. Your function will calculate the decay value of the filter using these inputs. (25pts)

- Using the provided code, look at the frequency impact of the filter in a white noise signal and answer the questions in the notebook. (25 pts)

4. Extended assignment (OPTIONAL, NO EXTRA CREDIT, JUST FOR FUN)

The following difference equation describes a slightly more complex high pass filter. It has one pole and one zero, and a steeper cutoff than the previous low pass filter. Similar to *d* in the low pass `filter`, *p* is also a constant between 0 and 1 that will determine the cutoff frequency of the filter. Implement the difference equation and test the effects of the filter on white noise and your audio file with different p values.

$$y[n] = x[n] - x[n-1] - p * y[n-1]$$