Lab 5: Filters

You are going to create two functions

filterSignal() – to run a specific type of filter on an inputted signal and returning the filtered signal

allFilters () – to call filterSignal() four times to run low-pass ('low'), high-pass ('high'), band-pass ('bandpass'), and band-stop ('bandstop') filters on an inputted signal and returning all of the filtered signals

1) In cell 1: import the necessary libraries

```
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
import IPython.display
import scipy.signal as sg
```

and the audio files from this GitHub repo

```
!git clone https://github.com/jcdevaney/ musicSignalProcessingBasics.git
```

2) In cells 2 copy plotTimeFreq() from filters.ipynb

3) In cell 3 create a function called filterSignal() that inputs

- signal to filter (sig)
- sampling rate of the signal (sr)
- filter frequency/frequencies (freq)
- filter type (filtType)
- order of the filter (order)
- window size for the FFT (winSize)
- type of spectrogram, log or linear (specType)
- figure number (fig)

The function will

- use an if/elif/else statement with 'or' tests to set up the parameters for the different type of filter based on the code in filters.ipynb and or.ipynb, specifically
    - if the filter type is 'low' or 'high'
        - filterFreq = freq / (sr / 2)
    - if the filter type if 'bandpass' or 'bandstop'
        - filterFreq = [freq[0] / (sr / 2), freq[1] / (sr / 2)]
- create a filter using sg.butter()
- run the created filter on the inputting signal using sg.filtfilt()
- plot the filtered signal with plotTimeFreq()

And it will return

- the filtered signal

4) In cell 4 create a function called allFilters() that inputs

- signal to filter (sig)
- sampling rate of the signal (sr)
- filter frequency (freq)
    - cutoff frequency for low-/high-pass
    - central frequency for band-pass/band-stop
- distance between the low and high band frequencies (width)
- order of the filter (order)
- window size for the FFT (winSize)
- type of spectrogram, log or linear (specType)

The function will

- call filterSignal() to run a low-pass filter on the inputting signal using the inputting filter frequency and filter order – fig = 1
- call filterSignal() to run a high-pass filter on the inputting signal using the inputting filter frequency and filter order, fig = 2
- call filterSignal() to run a band-pass filter on the inputting signal using the inputting filter frequency and filter order – filter frequency will be calculated using the inputted width parameter like this: [freq-width,freq+width], fig = 3
- call filterSignal() to run a band-stop filter on the inputting signal using the inputting filter frequency and filter order – filter frequency will be calculated using the inputted width parameter like this: [freq-width,freq+width], fig = 4

And it will return

- the four filtered signals (return sig_lp, sig_hp, sig_bp, sig_bs)

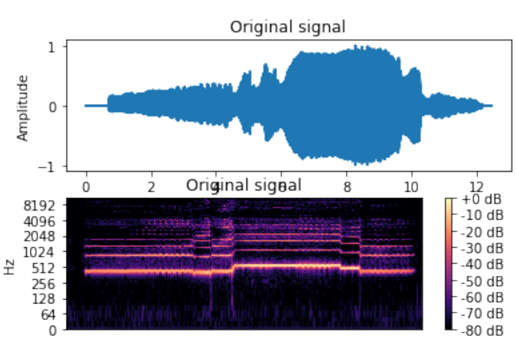
5) In cell 5 load, plot, and play the original signal

```
sig , sr = librosa.load('imc2023/audioFiles/avm.wav')
winSize = 1024
specType = 'log'
fig = 1

plotTimeFreq(sig,sr,'Original signal',winSize,specType,fig)

IPython.display.Audio(data=sig, rate=sr)
```

This should generate the following plot

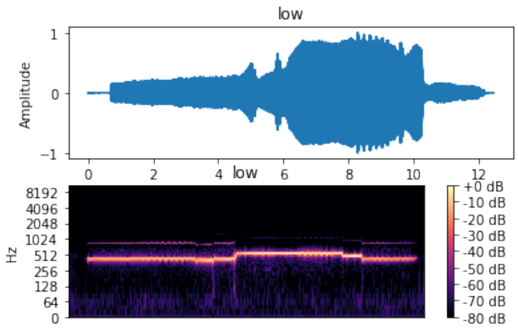


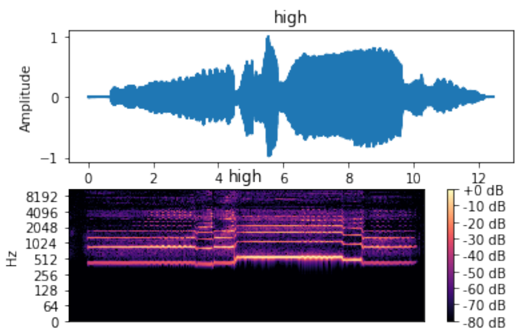
6) In cell 6: call allFilters () with the following arguments

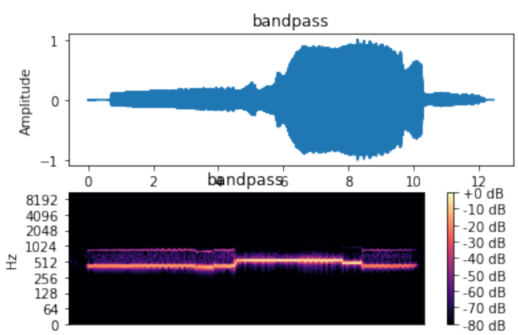
freq = 600
width = 200
order = 4

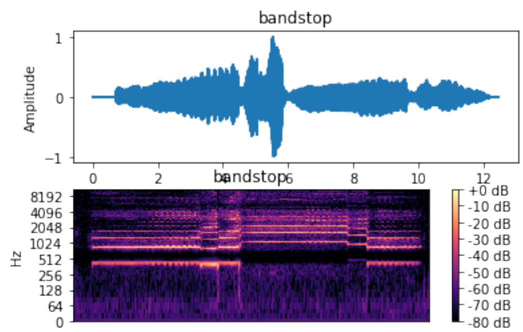
```
sig_lp, sig_hp, sig_bp, sig_bs=allFilters(sig,sr,freq,width,order,winSize,specType)
```

This should generate the following plots









6) In cell 6: play the low-pass version of the signal

```
IPython.display.Audio(data=sig_lp, rate=sr)
```

7) In cell 7: play the high-pass version of the signal

```
IPython.display.Audio(data=sig_hp, rate=sr)
```

8) In cell 8: play the band-pass version of the signal

```
IPython.display.Audio(data=sig_bp, rate=sr)
```

9) In cell 9: play the low pass version of the signal

```
IPython.display.Audio(data=sig_bs, rate=sr)
```