

## ENGR 362 DSP I: Guitar Pedal Filter Project

### Details.

Due date: April 4, 2022 at 11:59 pm.

Submission details: Submitted via the UBC Canvas website. You must submit a project report and a .m file.

Weight: 20% of your final mark.

All coding should be done in MATLAB and .m file must be uploaded to UBC Canvas website.

This is an individual assessment.

### Context.

Guitar pedals are used to modify (i.e., filter) the sound and frequencies of an electric or acoustic guitar. They are typically located on the ground, hence the name 'guitar pedal'. The signal is picked up by either a microphone of an electromagnetic pickup, then put through the guitar pedal(s), and then into the amplifier/speaker.



**Guitar pedals. (commons.wikimedia.org)**

These guitar pedals can be analog or digital. Digital guitar pedals can be designed using MATLAB and digital signal processing filters.

There are several common guitar pedals:

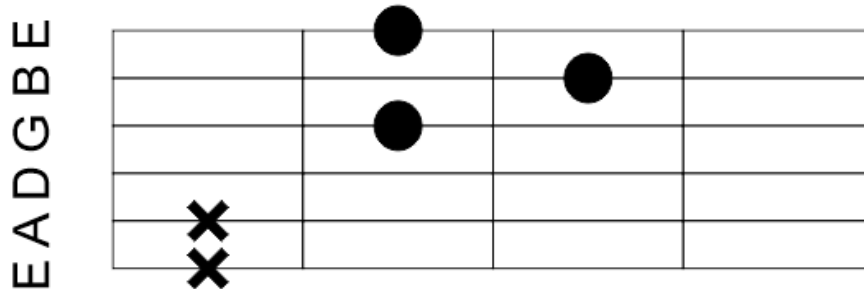
- i. Tuning pedal: Communicates how close to known pitches the guitar strings are tuned, i.e., it can discern if a string is tuned sharp (pitch too high frequency) or flat (pitch too low frequency)
- ii. Delay pedal: Creates echo effect by repeating signal in time-domain
- iii. Distortion pedal: Uses feedback to create a coarser sound
- iv. Compression pedal: This pedal compresses the volume (i.e., amplitude) of different pitches, i.e., making low volumes pitches higher volume and making high volume pitches a lower volume. This evens out the sound (with all amplitudes being close to equal) and is a popular effect for the guitar as many strings are played at once.

In this ENGR 362 DSP I project, students will use MATLAB to design the digital signal processing filters for a **digital guitar compressor pedal**.

## General description of DSP code.

You are to write a .m file DSP code to do the following:

- Input the text file *ENGR\_362\_guitar\_Fs\_is\_48000\_Hz.txt*
  - This is a recording of a guitar playing a D Major chord, which consists of four pitches: 146.83 Hz (D3), 220.00 Hz (A3), 293.66 Hz (D4), and 369.99 Hz (F#4)
  - This recording has sampling frequency of 48 kHz
- Play the audio of the file
- Output a graph of the time-domain signal
- Output a graph of the frequency-domain signal
- The signal should then go into a filter bank that you design. The filter bank should have four bandpass filters operating in parallel to isolate each of four above frequencies.
  - It is recommended that you construct your bandpass filters from two Chebyshev filters applied in series: one lowpass filter with cutoff frequency above the desired pitch and one highpass filter with cutoff frequency below the desired pitch
- Record in the code the amplitude of the filtered signals at the frequencies of the four notes. This will be needed to balance the amplitudes when recombining the signals
- Finally, adjust the amplitudes of the filtered signals to balance the amplitudes of the four frequencies. The amplitudes should be equal within 10% of each other; then recombine the signals and analyze in the frequency domain



**Fretboard notation for D major chord.**  
**The bottom two strings are muffled, while the top four strings form the pitches described above.**

## Hints.

A template .m file is given on the UBC Canvas website. This can be used to begin your code.

The MATLAB function `Cheby1` is recommended; it implements the Chebyshev filter (either highpass or lowpass). The syntax is as follows:

```
[b, a] = cheby1(n, Rp, Wp)
[b, a] = cheby1(n, Rp, Wp, 'high')
```

`b` and `a` are the difference equation coefficients

`n` is the order of the filter (experiment with this to see what works well)

`Rp` is the peak-to-peak bandpass ripple (experiment with this to see what works well)

`Wp` is the passband edge frequency

The last input can be 'high' or 'low' to indicate highpass or lowpass filter

```
y_filtered = filter(b, a, y);
```

This syntax will apply the filter corresponding to `b` and `a` difference equation coefficients (i.e., system function coefficients).

```
freqz(b,a)
```

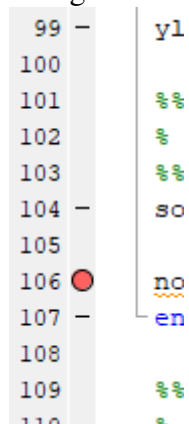
This can be used to view the frequency response of a filter. It can be used twice between `hold on` and `hold off` to plot two filters (e.g., highpass and lowpass) on the same graph.

```
soundsc(y,Fs)
```

This plays audio of a vector with an associated sampling frequency

A *for loop* can be used to implement the filter bank.

Using MATLAB debugging break points after each sound playback is helpful while working on your `.m` file.



The image shows a snippet of a MATLAB script in an editor. The script contains several comments and code lines. A red circle, indicating a break point, is set on line 106. The lines are as follows:

- 99 - `y1`
- 100
- 101 `%%`
- 102 `%%`
- 103 `%%`
- 104 - `so`
- 105
- 106 `no` (with a red break point circle)
- 107 - `en`
- 108
- 109 `%%`
- 110 `e`

## Report.

Your report should address the following:

- Provide a flow chart to explain how your code works
- Provide graphs of the original audio signal in the time and frequency domains
- Provide graphs of the frequency response of each of your bandpass filters
- Provide your four amplitudes by which to scale the four frequency amplitudes
- Provide a graph of the final compressed signal in the frequency domain, clearly showing that all amplitudes are within 10% of each other

## Marking.

Reports will be marked on the following:

- Flow chart logic
- MATLAB-generated plots and figures (must have all axes and data clearly labelled; must show correct and logical results)
- Justification of filter parameters (i.e., what led you to your chosen cutoff frequencies and other filter parameters?)
- Is the design within the specifications for compression? Why or why not?

- Appearance/presentation of the report
- Detailed descriptions and discussions of all presented data

.m file code will be marked on the following:

- Organization and documentation (i.e., comments) of the MATLAB code
- Logic of filter
- That the .m file works with the provided data text file