Class:	EE480L – Digital Signal Processing - 1001	Semester:	Fall 2022

Points		Document author:	Maxwell Stonham Edreese Basharyar Abraham Castaneda
		Author's email:	stonham@unlv.nevada.edu basharya@unlv.nevada.edu castaa7@unlv.nevada.edu
		Document topic:	Final Report

Instructor's comments:

1. Introduction

Our project was inspired by the use of pedals in playing the electric guitar. The three of us play the guitar in our spare time and after finishing Lab 9, we realized that MATLAB could be a cool tool to implement audio effects into our own music as a budget-friendly and quick way to add effects to prerecorded songs. Creating audio effects using MATLAB is not an uncommon idea, and we thought it would be interesting to use the techniques we learned throughout this lab to try and implement the most common audio effects we could think of, as well as the ones that are the easiest to notice for everyday listeners. We decided to go with one effect per member: a lowpass filter, echo effect, and flange effect.

2. Description of the Project

Group Member Roles

- Edreese Basharyar: Low Pass Filter implementation to filter out noise
- Abraham Castaneda: Flange effect implementation
- Maxwell Stonham: Echo effect implementation

Low-Pass Filter

One of the most influential concepts in digital signal processing is the design of low-pass filters. Low-pass filters can be used to remove high frequency noise from signals. This is because high frequency noise is often undesirable in many applications, as it can cause distortion or interfere with the desired signal. By using a low-pass filter, the high frequency noise can be removed or reduced, leaving the desired low-frequency signal untouched. This feature is very beneficial to any audio signal we use that may include any sort of noise or interference. However, there are some slight drawbacks that may change the properties of a signal that may not be wanted. While low-pass filters can be useful for removing high frequency noise, they can also cause some loss of quality in the signal.

This is because low pass filters work by attenuating or blocking high frequency signals, which can also include some of the desired high frequency components of the original signal. As a result, the output signal may have a slightly lower quality than the original, especially if the filter has a steep cutoff or a high attenuation level.

Echo Effect

The echo effect is a very common effect that is caused by the repetition of an input signal at a (normally) lower level. The echo effect can be described in the simple equation below:

$$y(n) = x(n) + a*x(n-d)$$

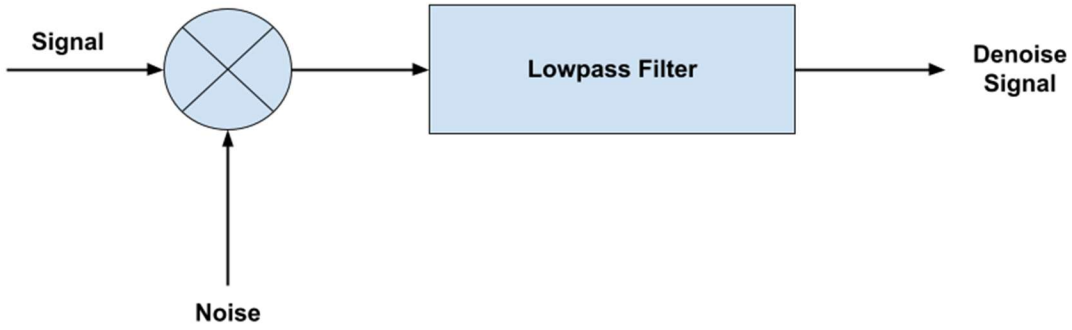
Where x(n) is the input audio, "a" is the gain or decay, x(n-d) is the shifted input audio and y(n) is the output signal. The equation describes the echo effect as the input signal added with a decayed and delayed version of that same input signal. This echo effect can be implemented on MATLAB either using convolution or without convolution.

Flange Effect

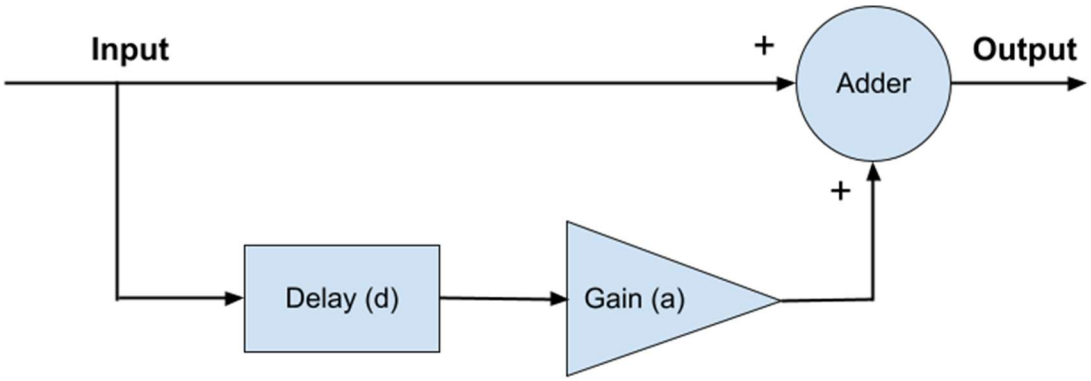
The flange sound effect is achieved by taking a signal, duplicating it, delaying the duplicated signal by a small amount, and varying the delay in a sinusoidal manner, after which the original and duplicated signal are combined. The variation in the delay difference among the combined signals creates an interference pattern in the sound that results in certain frequencies being attenuated while others are amplified as the signal plays. The flange effect is often used in rock music to add depth to guitar and drum tracks.

3. Project Function Breakdown Diagrams

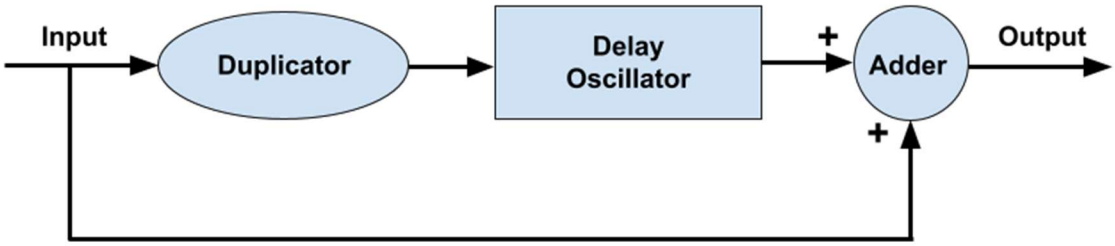
Lowpass Filter



Echo



Flange



4. Descriptions, Explanations (schematics and simulations) and Results

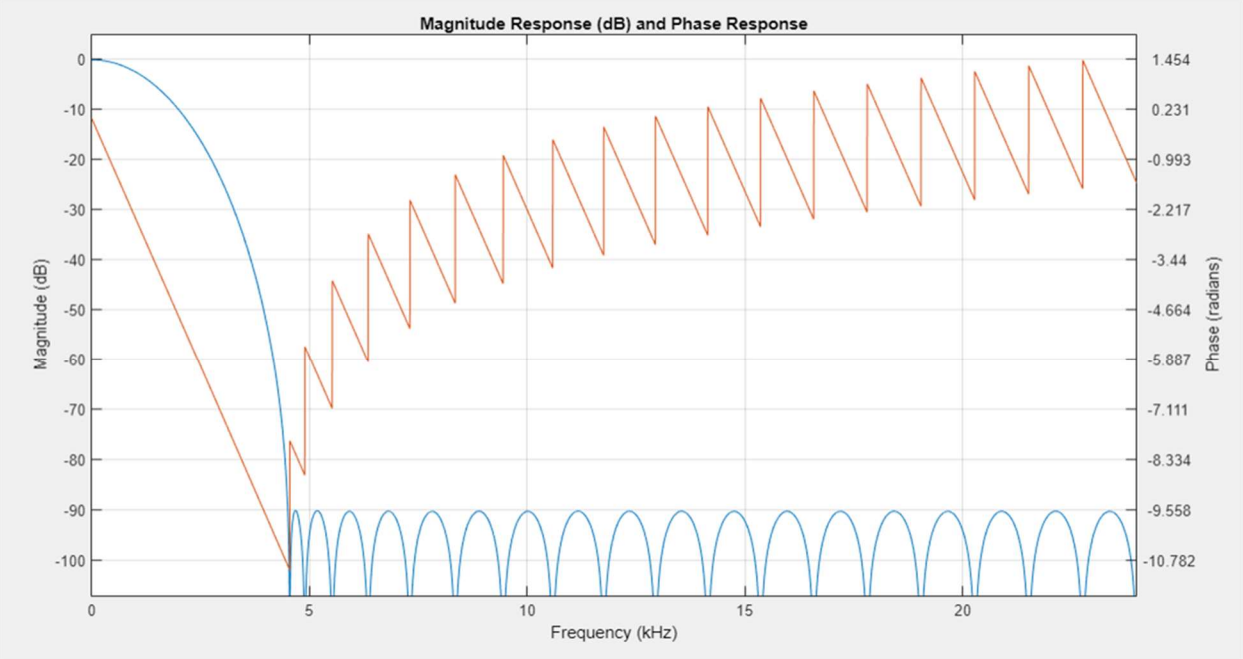
Low-Pass Filter

Implementation of Low-Pass Filter with MATLAB code:

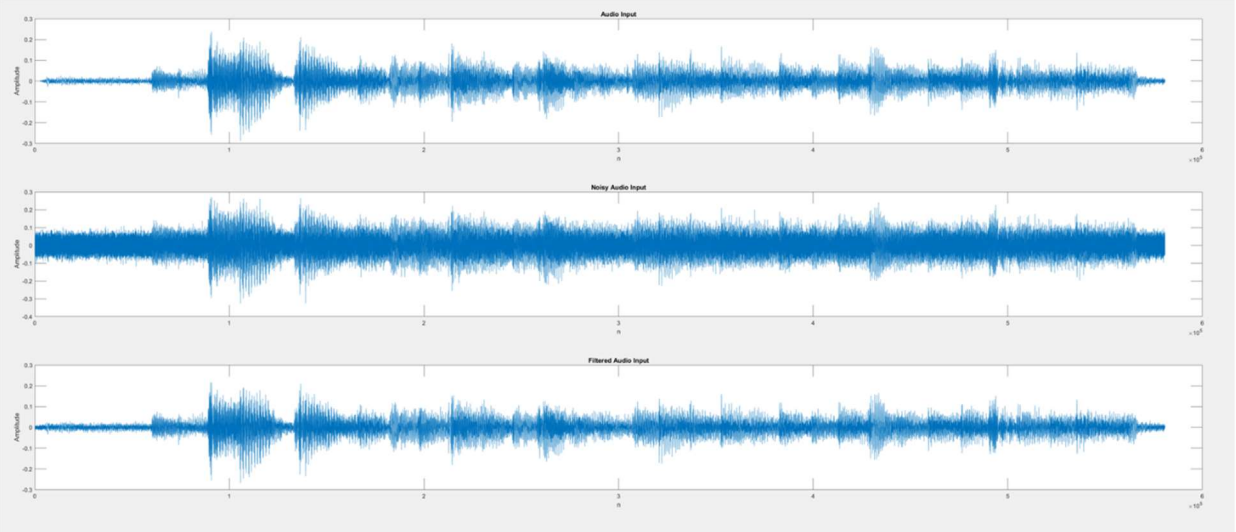
```
%% Low Pass Filter
clc
clearvars

% Read the audio file and store sample and sampling rate
[x,Fs] = audioread('k.mp3');
xn = awgn(x,4,'measured');                    % Add some white gaussian noise
X = fft(xn);                                  % Fourier transform of the signal
Xmag = abs(X);                                % Magnitude of the Fourier transform
f = (0:length(Xmag)-1)*Fs/length(Xmag);       % Frequency Range

figure(1)
plot(f, Xmag);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('FFT of Audio Signal')

% LPF Parameters
Fpass = 400;
Fstop = 4500;
Apass = 1;
Astop = 90;
% LPF Implementation
lpf = designfilt('lowpassfir', 'PassbandFrequency', Fpass, 'StopbandFrequency',...
Fstop, 'passbandRipple', Apass, 'stopbandAttenuation', Astop, 'sampleRate', Fs);
y = filter(lpf, xn);

% fvtool(lpf); % visualize freq response of filter
figure(2)
subplot(3,1,1)
plot(x)
xlabel('n')
ylabel('Amplitude')
title('Audio Input')

subplot(3,1,2)
plot(xn)
xlabel('n')
ylabel('Amplitude')
title('Noisy Audio Input')

subplot(3,1,3)
plot(y)
xlabel('n')
ylabel('Amplitude')
title('Filtered Audio Input')
```

MATLAB FIGURES

```
% LPF Parameters
Fpass = 400;
Fstop = 4500;
Apass = 1;
Astop = 90;
```



From the above figure, we can see the visualization of the low-pass filter design we have created; this custom filter is specifically tuned to the LPF parameters we have set in the figure above. The first parameter, Fpass, indicates the frequency at the start of the pass band. The second parameter, Fstop, indicates the frequency at the end of the stop band. Based on the figure above, we can see how the blue waveform begins to attenuate right at the value of Fstop. The next parameter, Apass, indicates the amount of ripple allowed in the passband, where the units are denoted in decibels. Since the value of Apass is very small, the ripple can only be seen by zooming in closer to the graph. The last parameter we have defined in this graph is Astop, which indicates the attenuation in the stop band in the form of decibels. Since we defined the value of Astop as 90, we can observe how the blue waveform attenuates at a peak value of ninety decibels. Therefore, we can observe how the LPF parameters match the properties of the low-pass filter waveform.



In the figure above, we can observe three different waveforms that indicate the different phases of the audio signal. The waveform at the top indicates the original signal after it was read into MATLAB. The waveform in the middle indicates the audio signal that is altered with the addition of white gaussian noise, a built-in function in MATLAB software. We can observe how the amplitude of the waveform is much higher overall compared to the original audio signal. The waveform at the bottom indicates the noisy audio signal after it has been passed through the low-pass filter. We can see a drastic difference in amplitude compared to the middle waveform; we can also see that the filtered waveform is very similar compared to its original audio signal. Although the filtered audio signal does not have the exact amplitudes as its original, we can see that it has succeeded in removing the white gaussian noise.

Echo Effect

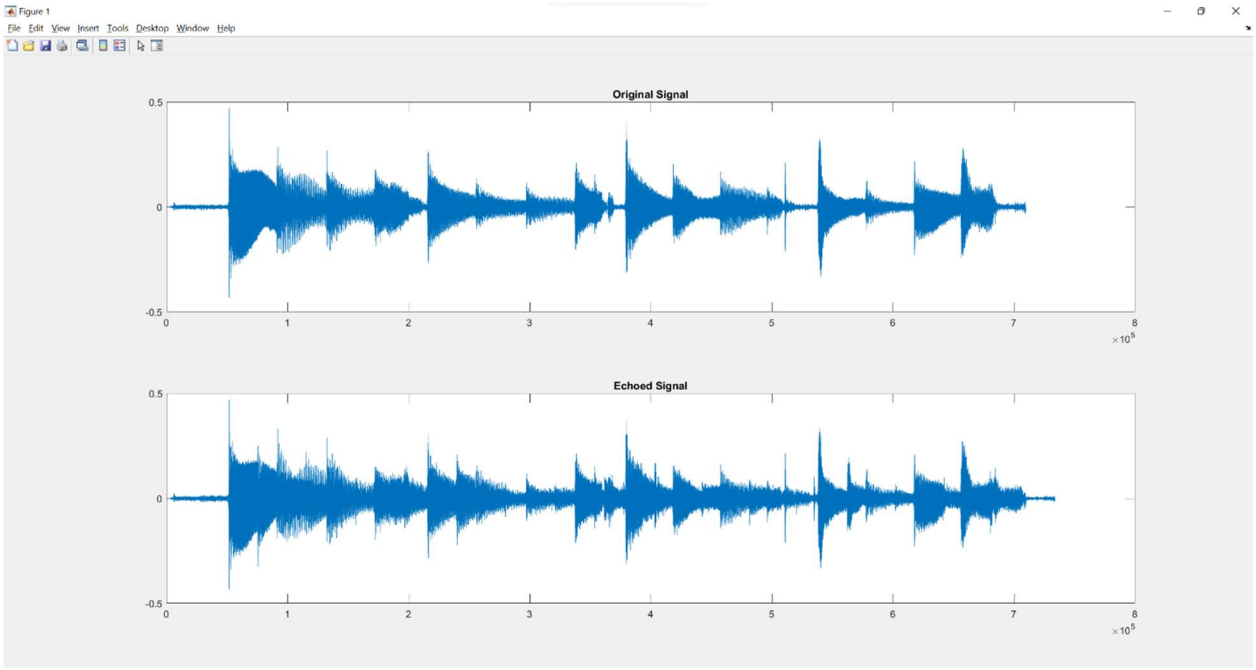
Implementation using convolution with a 50% delay and x0.5 gain:

```
%% Echo Effect
clc
clearvars

% Read in the mp3 file of choice
[x,fs] = audioread('forbidden.mp3');

d = fs/2;   % Set the delay value
a = 0.5;      % Set the gain/decay

% The delayed impulse response
h = [1, zeros(1,d), a];

% Convolve the audio signal with the impulse response
y = conv(x, h);

% Play the audio signal with the echo
sound(y, fs);
```

Workspace

Name	Value
a	0.5000
d	24000
fs	48000
h	*1x24002 double*
x	*709820x1 double*
y	*733821x1 double*

```
subplot(2,1,1)
plot(x)
title('Original Signal')
subplot(2,1,2)
plot(y)
title('Echoed Signal')
```

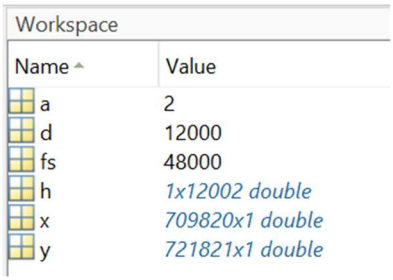


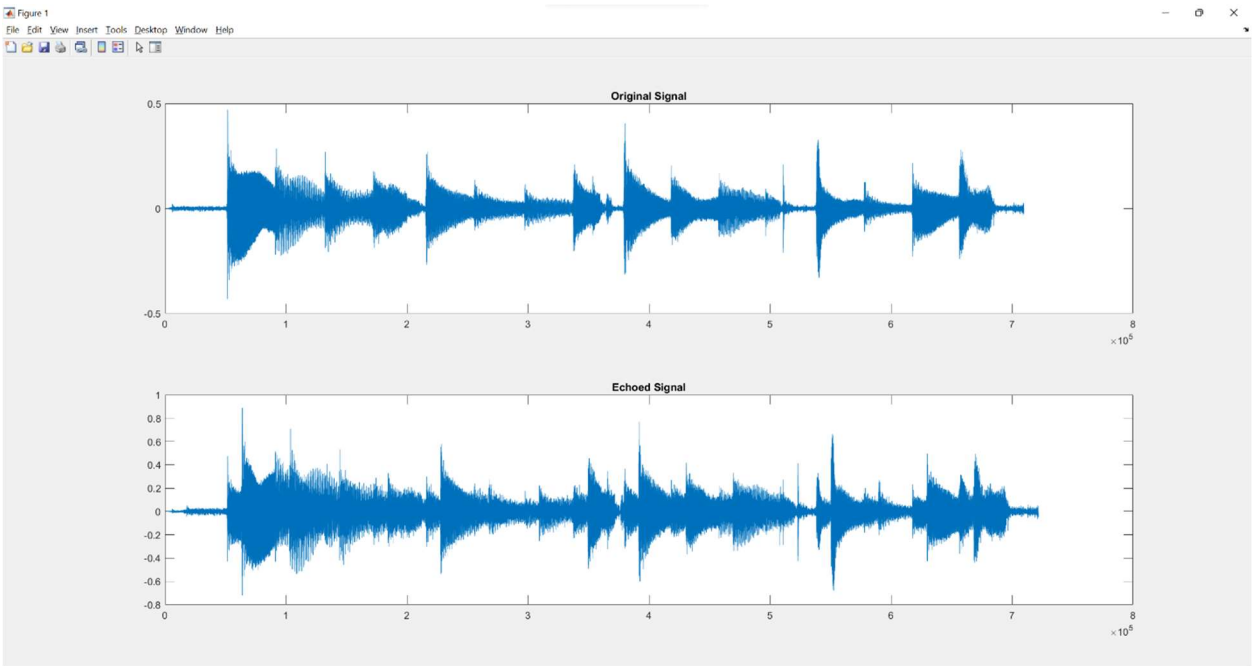
From the above code snippet and workspace, after reading in the forbiddem.mp3 files, we see that the original audio signal has a sample rate of 48000 that is stored in the variable “fs”. When we create our impulse response “h”, we will produce a response that is half of fs (24000) and store this value in “d” for our delay. We will also add a gain or decay in our variable “a” to output how loud we want our echo. Echoes are generally less loud than the input, so for the example above we will keep our decay at 0.5 times smaller than the input, and 0.5 times less than the original sample rate.

Shown in the above figure, we see the original signal visually represented on a graph. Below is the echoed signal implemented using the convolution code shown above. We see that the echoed signal is shown (notice the spikes and the difference between the graphs) to be half the amplitude of the original signal and half the time of the original (i.e., delayed by half the original signal). This is a successful representation of an echoed audio effect and can be modified depending on how the user would like to use the effect. It can either be delayed by a very short period, a very small echo audio that is barely heard, or even an echo signal that is much louder than the original. All this can be done by simply changing the “a” and “d” variables from the code snippet.

Implementation using convolution with a 25% delay and x2 gain:

```
%% Echo Effect
clc
clearvars

% Read in the mp3 file of choice
[x,fs] = audioread('forbidden.mp3');

d = fs/4;   % Set the delay value
a = 2;      % Set the gain/decay

% The delayed impulse response
h = [1, zeros(1,d), a];

% Convolve the audio signal with the impulse response
y = conv(x, h);

% Play the audio signal with the echo
sound(y, fs);

subplot(2,1,1)
plot(x)
title('Original Signal')
subplot(2,1,2)
plot(y)
title('Echoed Signal')
```

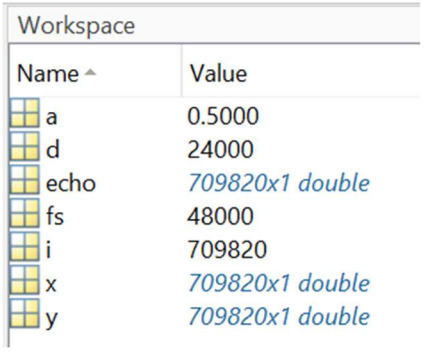


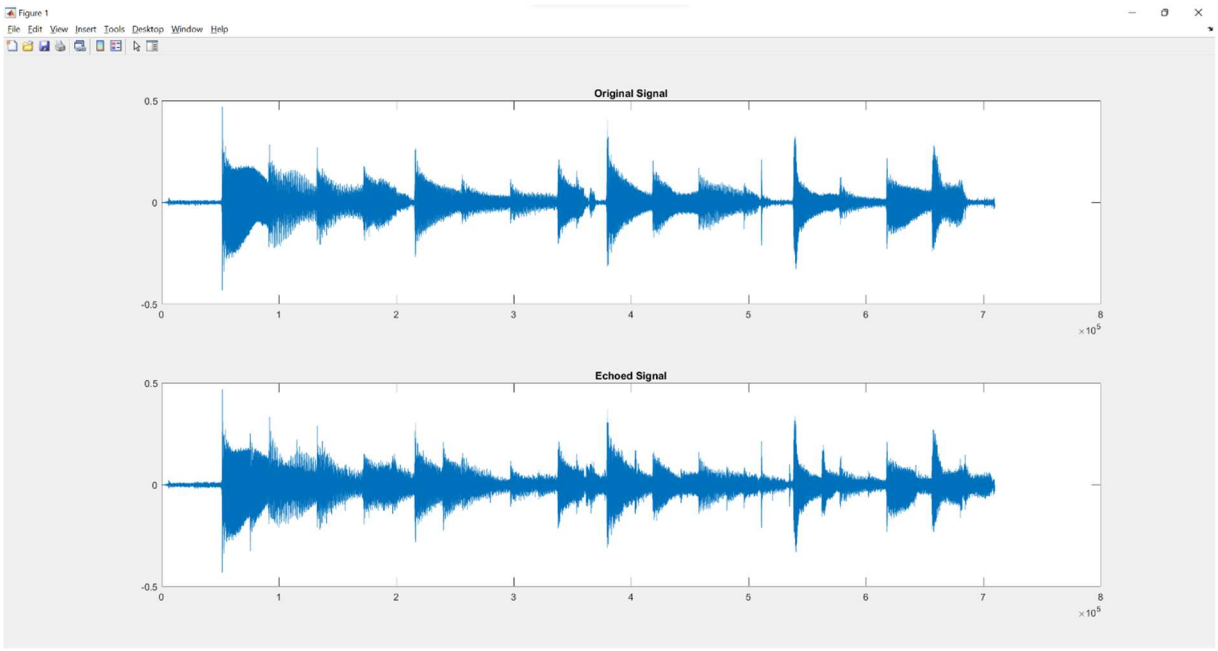


As a separate example, we see that modifying the gain and delay values does change the output echoed signal shown in the figure above. We see that setting our gain to 2 doubles the amplitude and that the echoed amplitude that is doubled is also echoing at 1/4th the sample rate of the original audio (i.e., from 48000 to 12000) as shown in the workspace. This shows that our code works to gain the echo at any rate the user would like as well as delay the sampling rate at any period.

Implementation without convolution with a 50% delay and x0.5 gain:

```
%% Echo Effect (Loop)
clc
clearvars

% Read in the mp3 file of choice
[x,fs] = audioread('forbidden.mp3');

d = fs/2;   % Set the delay value
a = 0.5;    % Set the gain/decay

% Set the length for variable echo
echo = zeros(size(x));

for i = 1:length(x)
    % Apply the delay by shifting the samples
    if i > d
        echo(i) = x(i-d)*a;
    end
end

y = x + echo;   % Add the echo with the input
sound(y,fs)    % Play the audio signal with the echo

figure(1)
subplot(2,1,1)
plot(x)
title('Original Signal')
subplot(2,1,2)
plot(y)
title('Echoed Signal')
```





We see that a similar result can be done without the use of convolution. Using the same variables for the delay and gain, this method simply uses a for loop and if statement to determine when the loop should start delaying the input signal and storing that signal into a variable "echo". This delay is done by having the loop only start after the variable "i" is greater than the set delay value chosen, in which case it will then start to fill in the initially empty "echo" length with the audio signal. This gives the same result as one would expect when doing it using convolution (see the peaks of the graph for the delays and amplitude), so it is a matter of preference.

Flange Effect

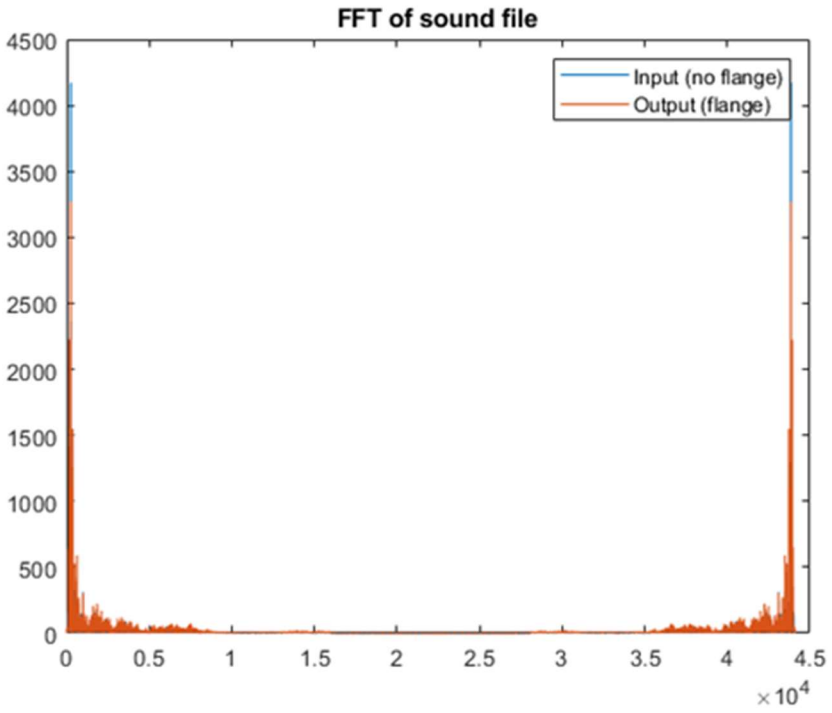
Implementation of flange in MATLAB (done to replicate the intro to "Barracuda" by Heart):

```
clc
clearvars

% Read audio file
[x, fs] = audioread('Barracuda.mp3');
Fx = fft(x,fs);

n = length(x);
tn=n/fs;

f = 0.21; %sets the frequency of the sinusoid that is delaying the signal
t = linspace(0,tn,n);
d = 100; %delay factor

modsin = sin(2*pi*f*t); %delay oscillator

modsin1 = round(d.*modsin') + d; % delay oscillator combined with delay
% factor
```

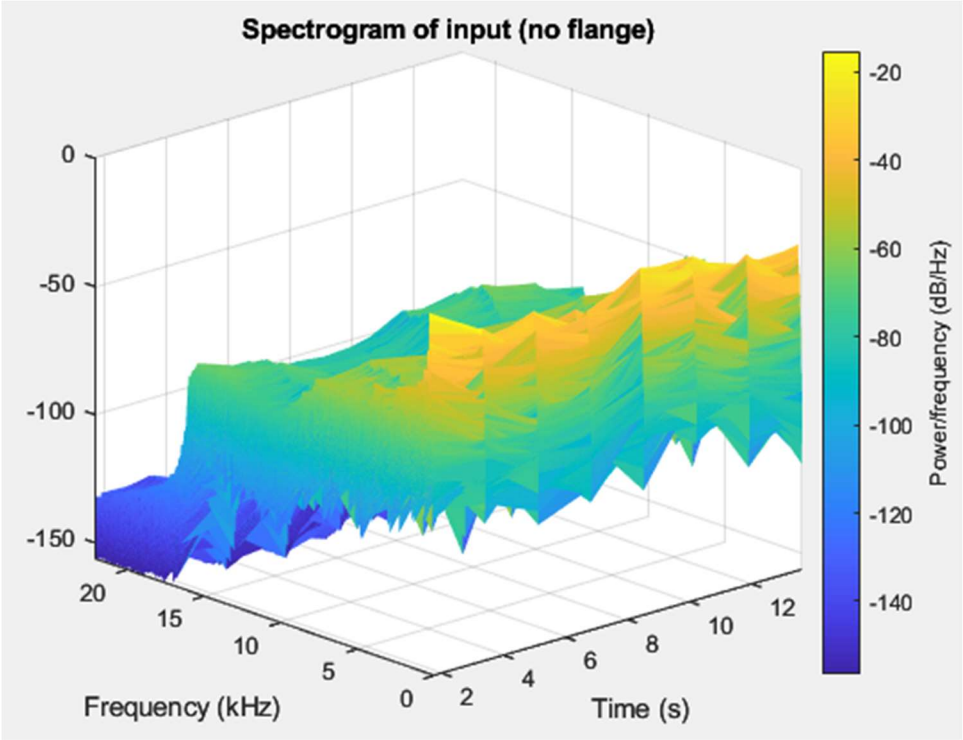
```
y = zeros(n+d,1); % output matrix prepared
a = 0.8; %attenuation factor

xn = padarray(x,[d,0],0,'pre'); % original signal matrix size padded to
% account for delay

% original signal combined with delayed signal
for i = (d+1):1:n
    y(i-d,1) = x(i) + a*xn(i-modsin1(i-d));
end

% flanged output
audiowrite('Barracuda_Flanged.wav',y,fs)
Fy = fft(y,fs);

figure(1)
plot(abs(Fx))
hold on
plot(abs(Fy))
title('FFT of sound file')
legend('Input (no flange)','Output (flange)')

figure(2)
spectrogram(x,[],[],[],fs,'yaxis')
title('Spectrogram of input (no flange)')

figure(3)
spectrogram(y,[],[],[],fs,'yaxis')
title('Spectrogram of output (with flange)')
```

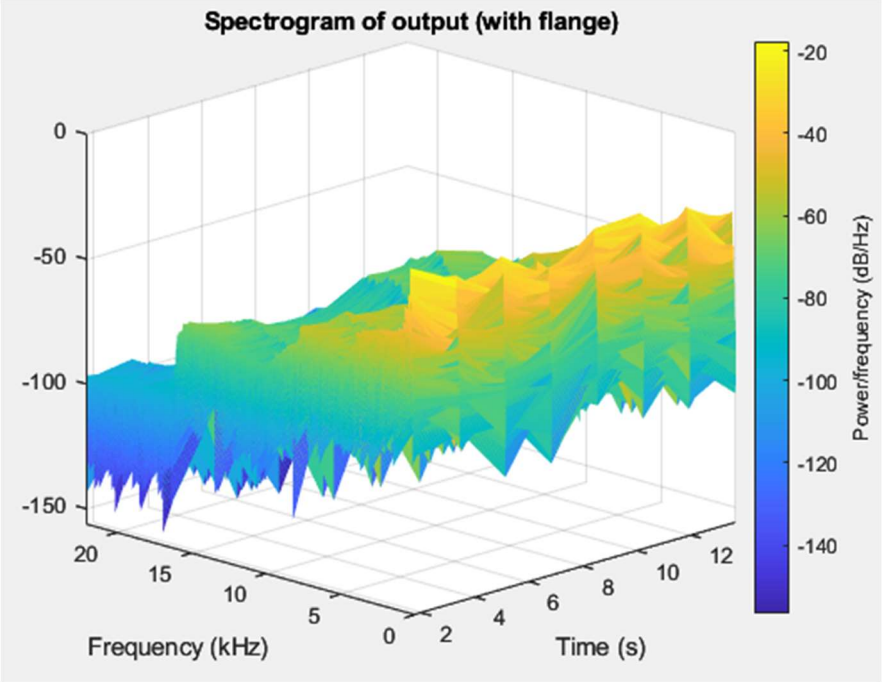
As can be seen from the code above, an input sound sample (in this case a guitar riff) is read into MATLAB, after which the delay parameters such as the delay factor and the delay oscillator frequency are set. The combined delay parameters are encoded in the sinusoidal function **modsin1**, which is later used alongside the attenuation factor to modify xn, an appended copy of the original signal.

The frequency of the delay determines how quickly the "wooshing" effect takes place. The delay factor determines how much the two combined sound signals will end up cancelling each other out (should not be too large or else the flange will not work properly). The attenuation factor determines the final amplitude of the duplicate sound signal before it is combined with the original sound signal, as this factor is increased, the depth of the flange also increases noticeably. The parameters chosen in this specific code were chosen to match the flange settings of the original song as closely as possible, but they can be altered as the user pleases.



Pictured above are the Fast Fourier Transforms of both the unflanged input and the flanged output. While there is not a large amount of difference, we can see that there is at least some cancellation of the very large spectrum peaks caused by the interference pattern from the flange signal combination process.





Pictured above are the spectrograms of the unflanged input and the flanged output. This time, the difference between the input and output are more noticeable than they were in the FFT chart, as the negative power spectral spikes due to the interference pattern are visible in the second plot, while in the first plot they are much smaller in number and in amplitude.

5. Problems Encountered

As far as problems are concerned, the main problem we encountered was the implementation of the equalizer. Initially, we planned to do an equalizer as one of our 3 demonstrations. An equalizer is a tool that adjusts audio signals based on the frequencies that the user would like to emphasize to shape the overall tone of the audio to emphasize certain frequencies and deemphasize others. In other words, an equalizer can boost specific frequencies, reduce them, or cut them out entirely. We were having trouble implementing this due to our time constraint, so we ended up doing the next most similar thing which was the lowpass filter implementation. The difference between the two would be that an equalizer can add or remove frequencies from an input signal, a lowpass filter simply filters out higher frequencies and only allows lower frequencies to pass. On top of this time constraint, seeing that we had labs implementing the lowpass filter, we thought it would be more fitting to add this instead since it is more closely related to what we learned in class.

6. Conclusions

In conclusion, we observed how audio effects can be implemented to alter the properties of audio signals in several ways with the use of MATLAB software. With the inspiration of playing the guitar, we created a low-pass filter, an echo effect, and a flange effect that can implement the same functionalities as other expensive audio-effect devices can provide. The three effects we have chosen are abundantly used in most audio signals, and our creation allows individuals to use these effects in a budget-friendly way. Such practices, while applicable to hobbyists, are often employed in music post-production to enhance the quality and marketability of the finished product, which is a testament to digital signal processing's very wide range applications.