# SIGNAL PROCESSING PROJECT REPORT

# MONSOON 2023

**TEAM: UNFILTERED**

Kushang Agarwal – 2022102076
Sriram Alluri – 2022102016
Sreevidhu Rithwik Yarra – 2022102012

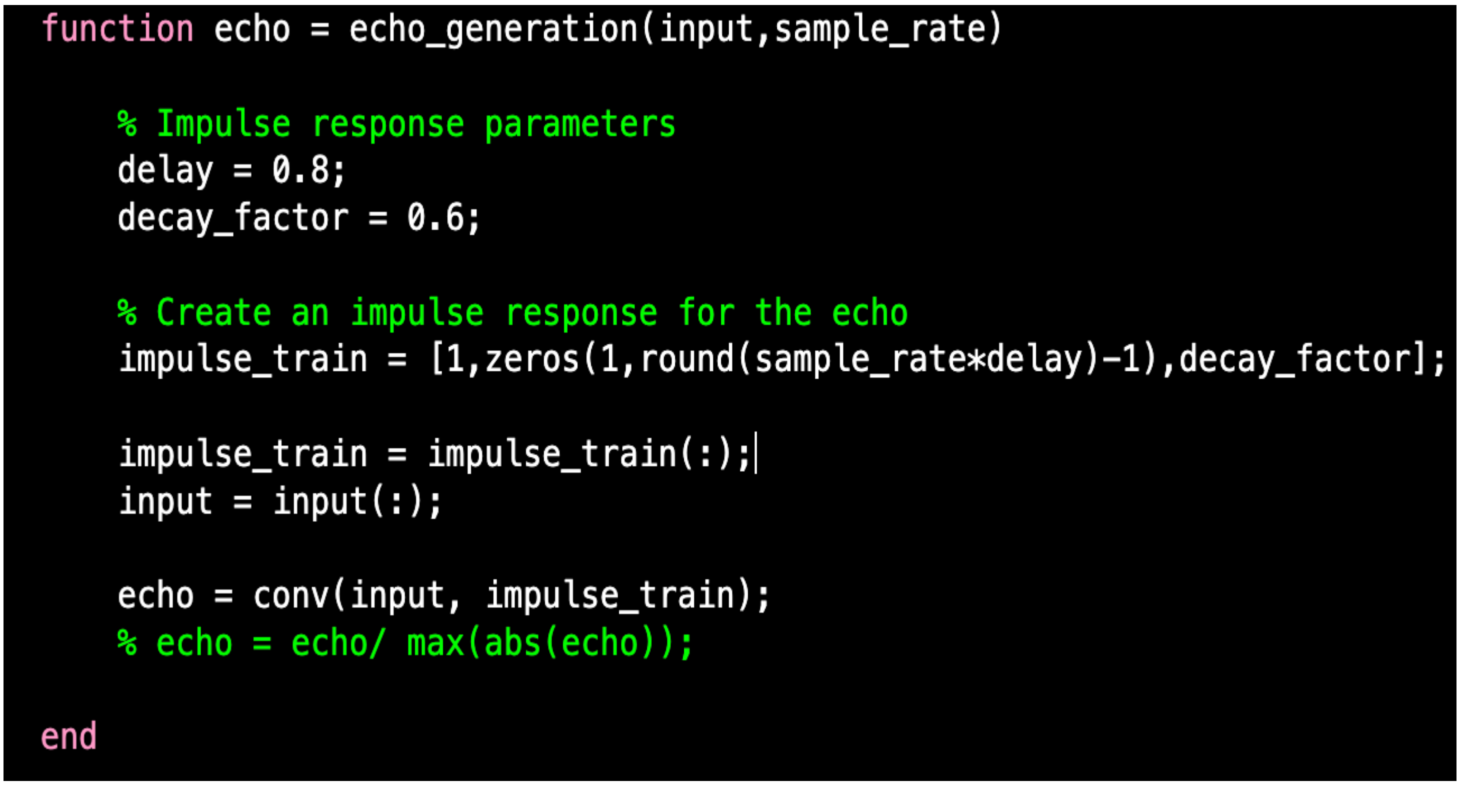
## Part 1 - Echo creation

Aim: - To apply signal processing techniques to create an echo effect for the given audio

Input: - A clean audio file without echo in .wav format.

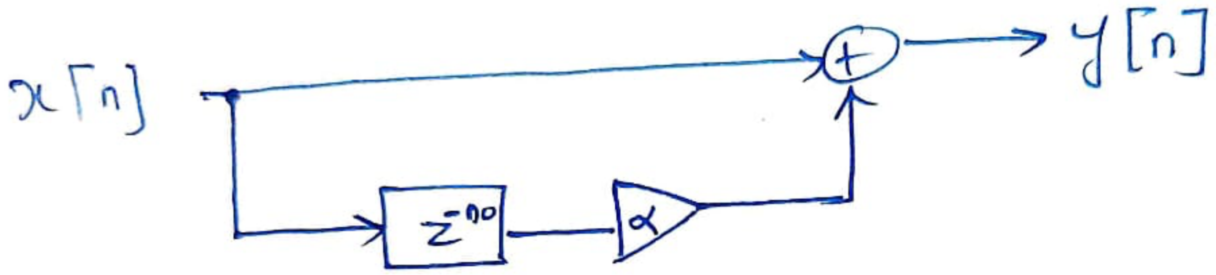
Problem solving approach: -

This is the function that we created to implement the task of creating an echo for the input audio file given.



```
function echo = echo_generation(input,sample_rate)

    % Impulse response parameters
    delay = 0.8;
    decay_factor = 0.6;

    % Create an impulse response for the echo
    impulse_train = [1,zeros(1,round(sample_rate*delay)-1),decay_factor];

    impulse_train = impulse_train(:);
    input = input(:);

    echo = conv(input, impulse_train);
    % echo = echo/ max(abs(echo));

end
```

The block diagram for the implementation of this approach is : -



Here, alpha is the attenuation factor and n0 is the delay introduced to produce the echo effect.

Let’s break down the algorithm step by step: -

Delay: - This parameter determines the time delay for the echo, specified in seconds.

Decay factor: - It represents the attenuation factor for the echo.

Impulse Train: - We create an impulse train based on the block diagram modelled to simulate the echo effect. A unit impulse is placed at the beginning of the response. Zeros are padded to simulate the delay, based on the specified ‘delay’ and ‘sample_rate’. The decay factor is applied at the end of the impulse response to model the diminishing amplitude of the echo over time.

We convolute this impulse train with the input audio signal and the resulting audio produces the echo effect. We can also normalize the output after the convolution step to prevent clipping of the output echo. Clipping can introduce distortion and artifacts into the audio signal.

The parameters **delay** and decay_factor can be adjusted to control the characteristics of the echo.

Why Convolution Works?

The convolution of two functions f(t) and g(t) is defined as: -

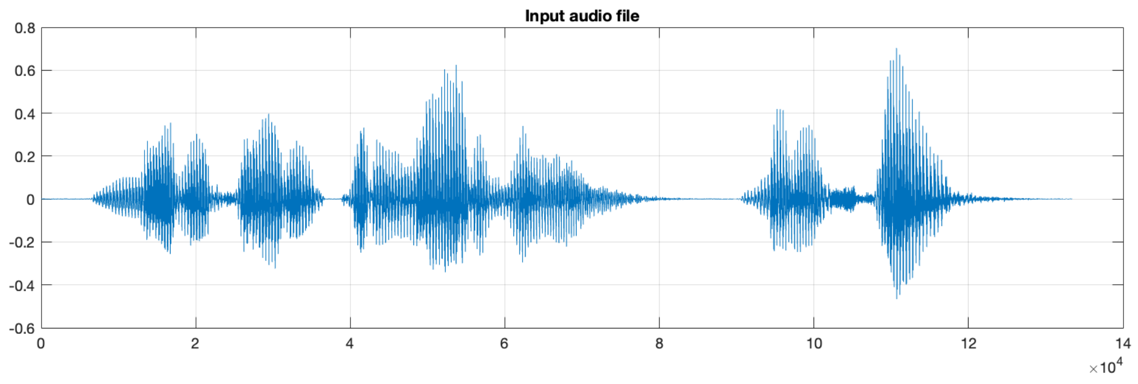
$$(f * g)(t) \overset{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

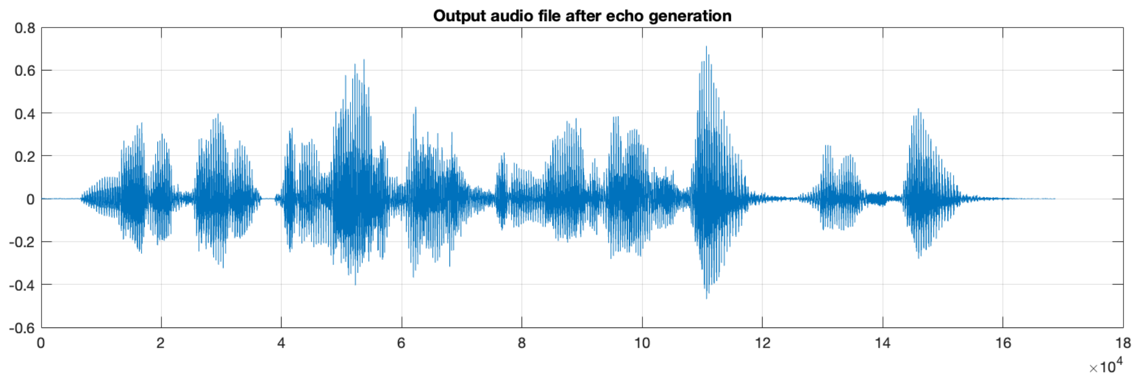
This integral represents the area under the product of the two functions as one slides over the other. In the time domain, convolution operation can be thought of as placing a copy of g(t) at each point in time and integrating the product with f(t).

The linearity property of convolution is the way to go about the proof. The response to a sum of signals is the sum of individual responses. In the case of an echo, the original signal and the delayed/attenuated version are convolved, and the resulting signal is the sum of these individual responses.

## Results: -

The resulting echo file is in the Output Audio/Plots folder with the name ‘output_q1_hindi_2s.wav’.





## Conclusion: -

In summary, convolution works for echo generation because it effectively models the time-domain interaction between an input signal and a system characterized by an impulse train. The linearity property of convolution allows us to simulate the echo effect by convolving the input signal with the impulse train.

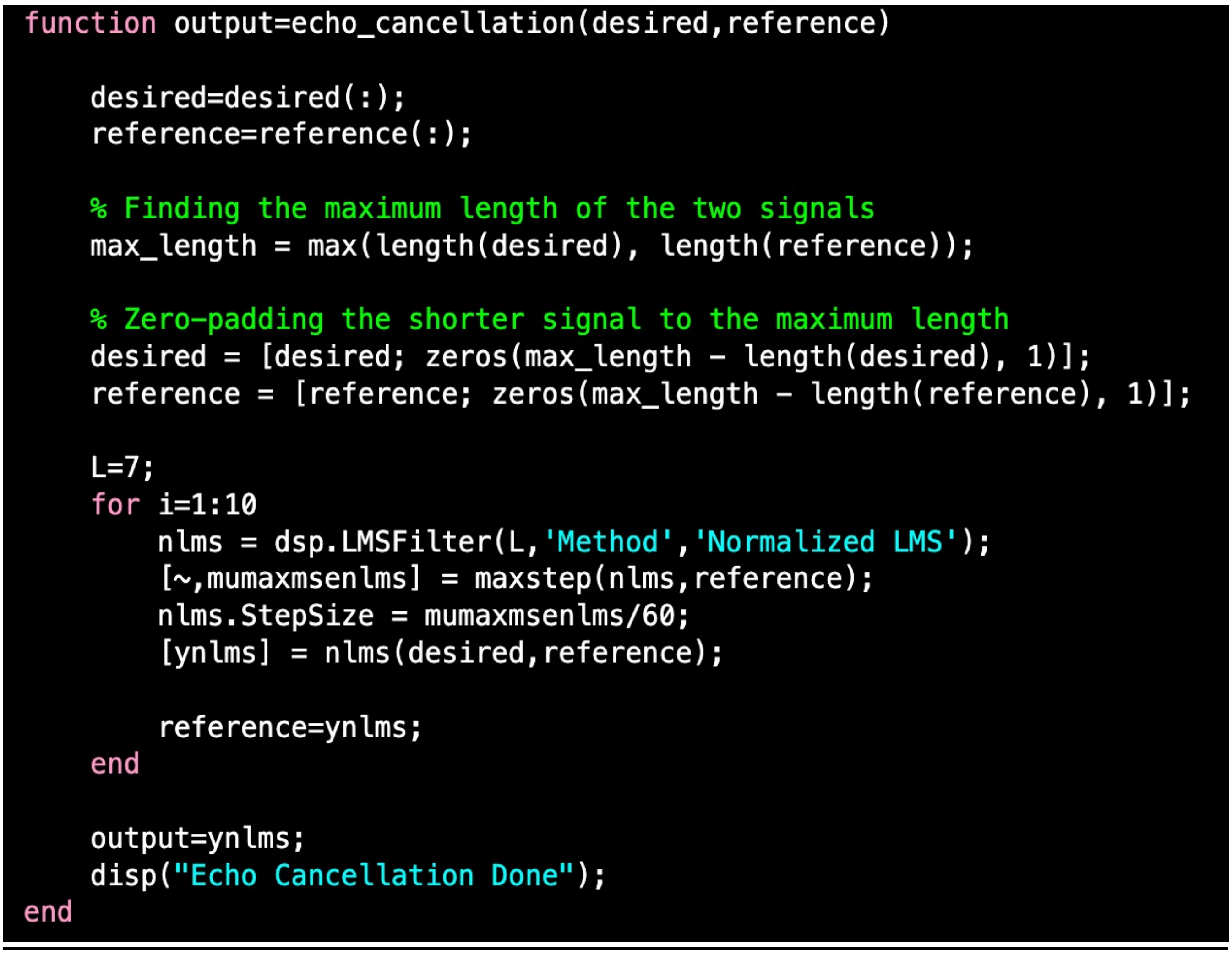
## Part 2 - Cancel the echo

Aim: - To apply signal processing techniques to cancel an echo effect for the given audio file.

Input: - Any audio file with echo in .wav format and the desired output as well.

Problem solving approach: -

This is the function that we created to implement the task of cancelling an echo for the input audio file given.



```
function output=echo_cancellation(desired,reference)

    desired=desired(:);
    reference=reference(:);

    % Finding the maximum length of the two signals
    max_length = max(length(desired), length(reference));

    % Zero-padding the shorter signal to the maximum length
    desired = [desired; zeros(max_length - length(desired), 1)];
    reference = [reference; zeros(max_length - length(reference), 1)];

    L=7;
    for i=1:10
        nlms = dsp.LMSFilter(L,'Method','Normalized LMS');
        [~,mumaxmsenlms] = maxstep(nlms,reference);
        nlms.StepSize = mumaxmsenlms/60;
        [ynlms] = nlms(desired,reference);

        reference=ynlms;
    end

    output=ynlms;
    disp("Echo Cancellation Done");
end
```

Let’s break down the algorithm step by step: -

- *Input Parameters*: We take two vectors as input (**desired** and **reference**).
- *Vectorization of Inputs*: Then, we ensure that desired and reference are column vectors.
- *Zero-padding*: We zero-pad the shorter signal to make both signals of equal length.
- *Initialization*: We set the filter length (L) to 7.

- *Adaptive Filtering Loop* (NLMS):
    - We create an LMS filter using the DSP System Toolbox in MATLAB.
    - Then, we set the method to "Normalized LMS."
    - We determine the maximum step size for stability using the maxstep function and use this as the step size for the NLMS Filter.
    - Then, we update the filter coefficients after every loop using the NLMS algorithm.

- *Output*:
    - The output of this function will be the final NLMS-filtered signal.

Why NLMS Filtering Works?

The Normalized Least Mean Square (NLMS) algorithm's effectiveness for echo cancellation can be attributed to its adaptive nature, which involves adjusting the filter coefficients based on the power of the input signal. The update equation for the weight vector in the NLMS algorithm is given by:

$$\mathbf{w(n+1) = w(n) + (\alpha / (c + x(n)^\wedge T * x(n))) * e(n) * x(n)}$$

where:
- **w(n)** is the weight vector at iteration n
- **α** is the step-size parameter
- **c** is the normalized step-size
- **x(n)** is the input signal vector at time n
- **e(n)** is the error signal at time n

The error signal at time n in the NLMS algorithm is given by:

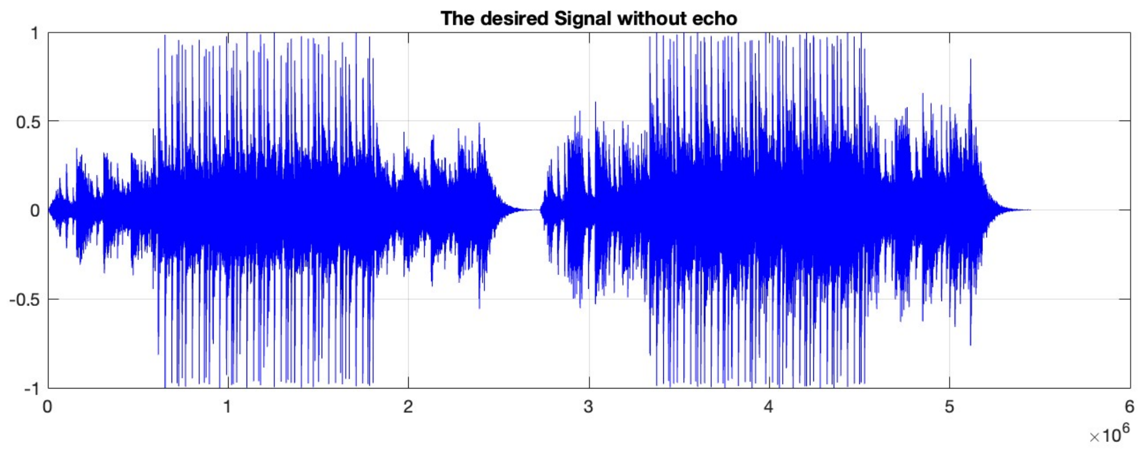
**e(n) = d(n) - y(n)**

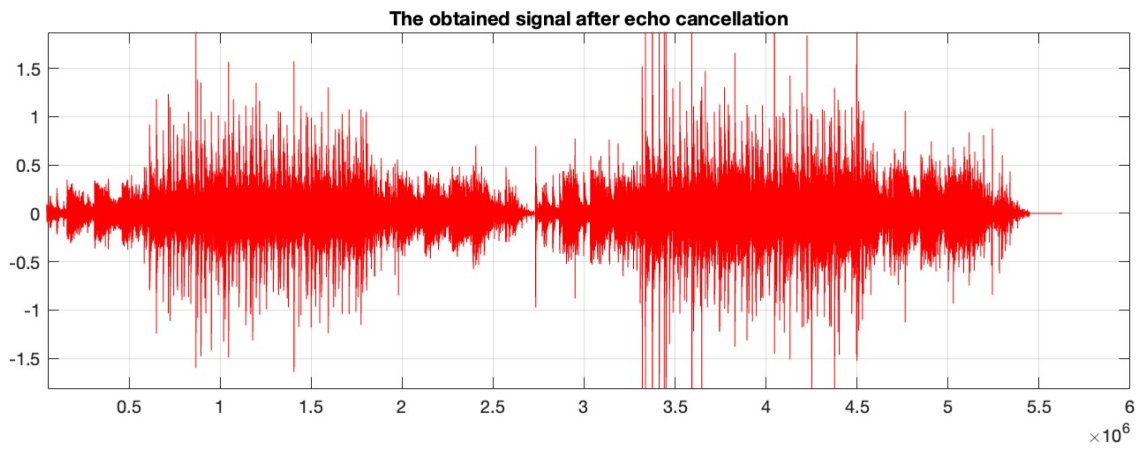
Where:
- **d(n)** is the captured microphone signal sample at time n
- **y(n)** is the output signal at time n, which is the inner product of the weight vector and the input signal vector at time n

The normalization of the step-size parameter with the input signal power allows the NLMS algorithm to adapt to changing acoustic environments and accurately estimate and remove the echo, leading to improved echo cancellation performance in telecommunication systems.

<u>Results: -</u>

We test our function using the echo created in q1 for the 'q1_hard.wav' file for city traffic. The resulting echo file is in the Output Audio/Plots folder with the name 'output_q2.wav'.

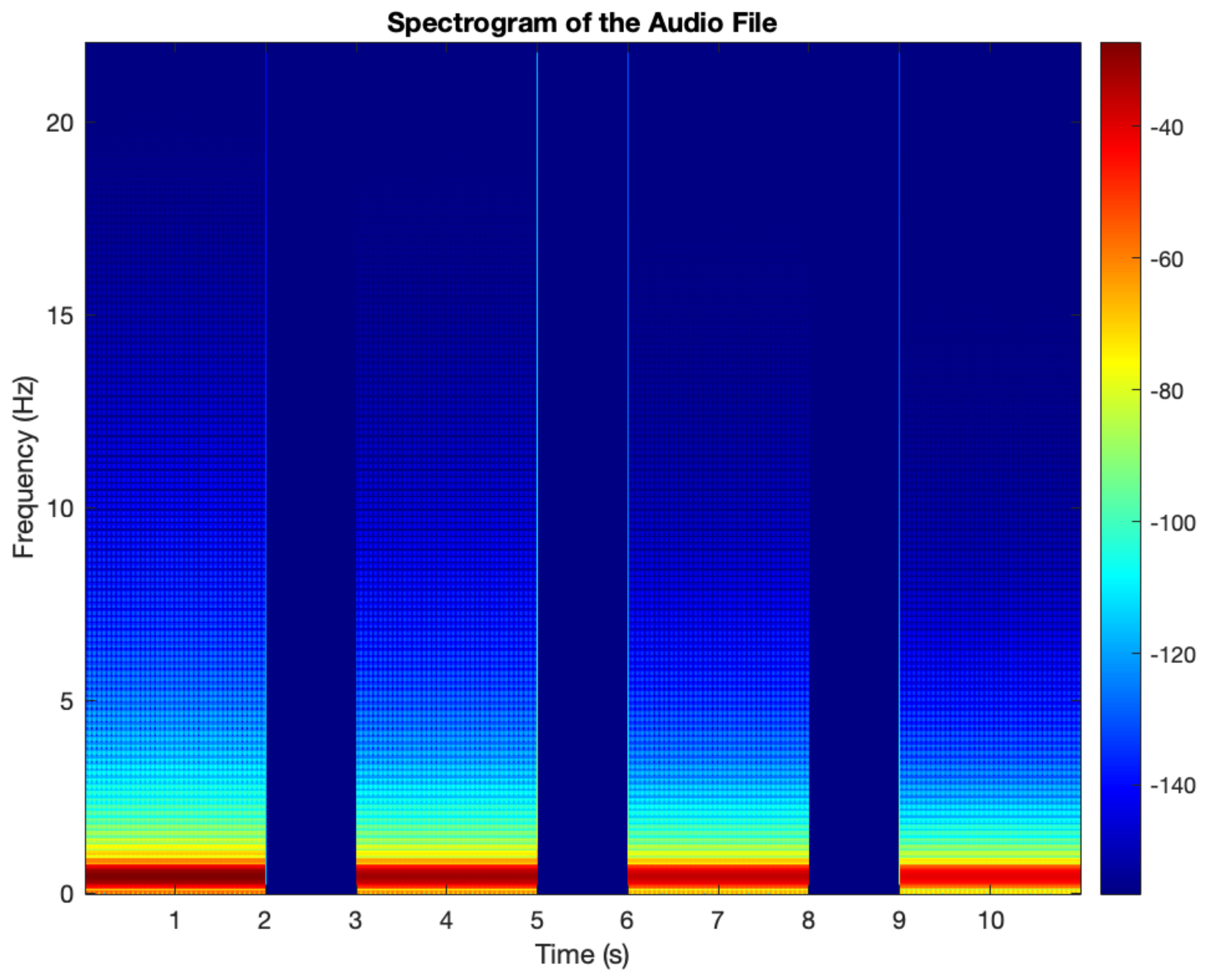




<u>Conclusion</u>:

So this method doesn't really work properly in real life since we don't have the signal of desired output beforehand. This adaptive filtering technique is more useful for stuff like system identification.

Another approach to this problem that we came up with was that we use spectrographs to look at the signal's amplitude and frequency at the same time. We can identify the echo's manually and use FFT algorithm to convert the signal in the frequency domain and then cancel out the extra parts of the echo and repeat the process for the rest of the signal. A spectrograph of 'q2_easy.wav' is shown below: -



However, we couldn't come up with a proper implementation for this technique and thus couldn't present this but we think this is a viable approach for echo cancellation.

## Part 3: What is this noise?

Aim: - To accurately identify and categorise the type of noise present in the recording/ To classify background noise in music recordings without removing the noise.

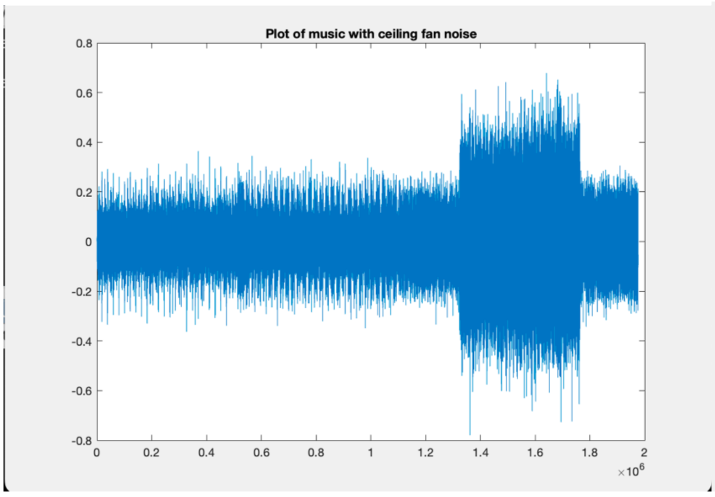
The type of noises given were: -

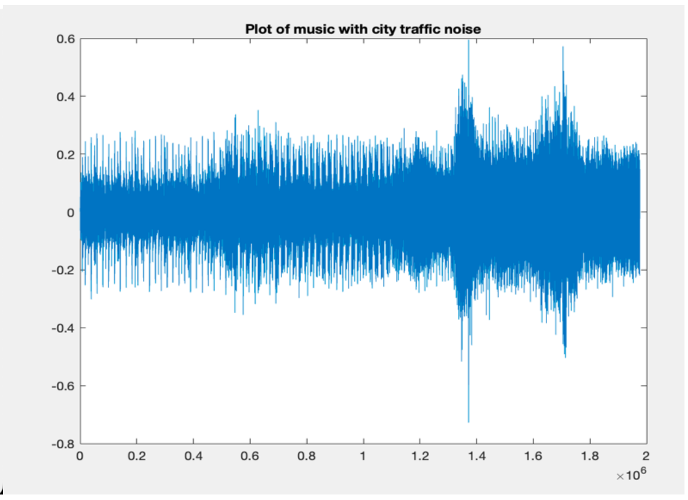
- ➔ Fan
- ➔ Pressure cooker
- ➔ Water Pump
- ➔ Traffic

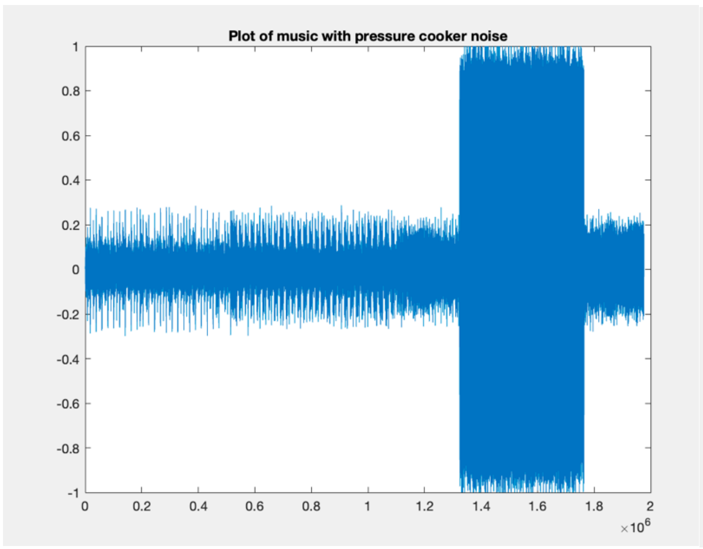
Input: - The input file is a music recording which contains both the original music and background noise from indoor and outdoor sources.

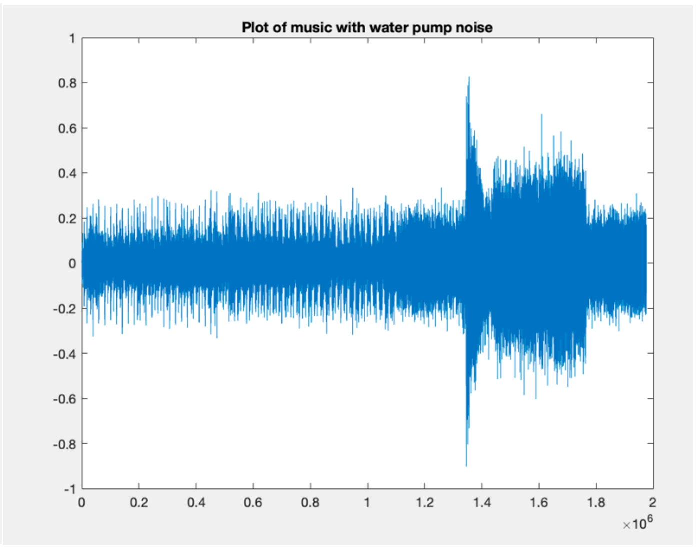
Problem solving approach: -

We were given 4 individual audio files initially which contained music along with the type of noises mentioned above. So, plotting the 4 audio files given along with the project gives us the following plots: -



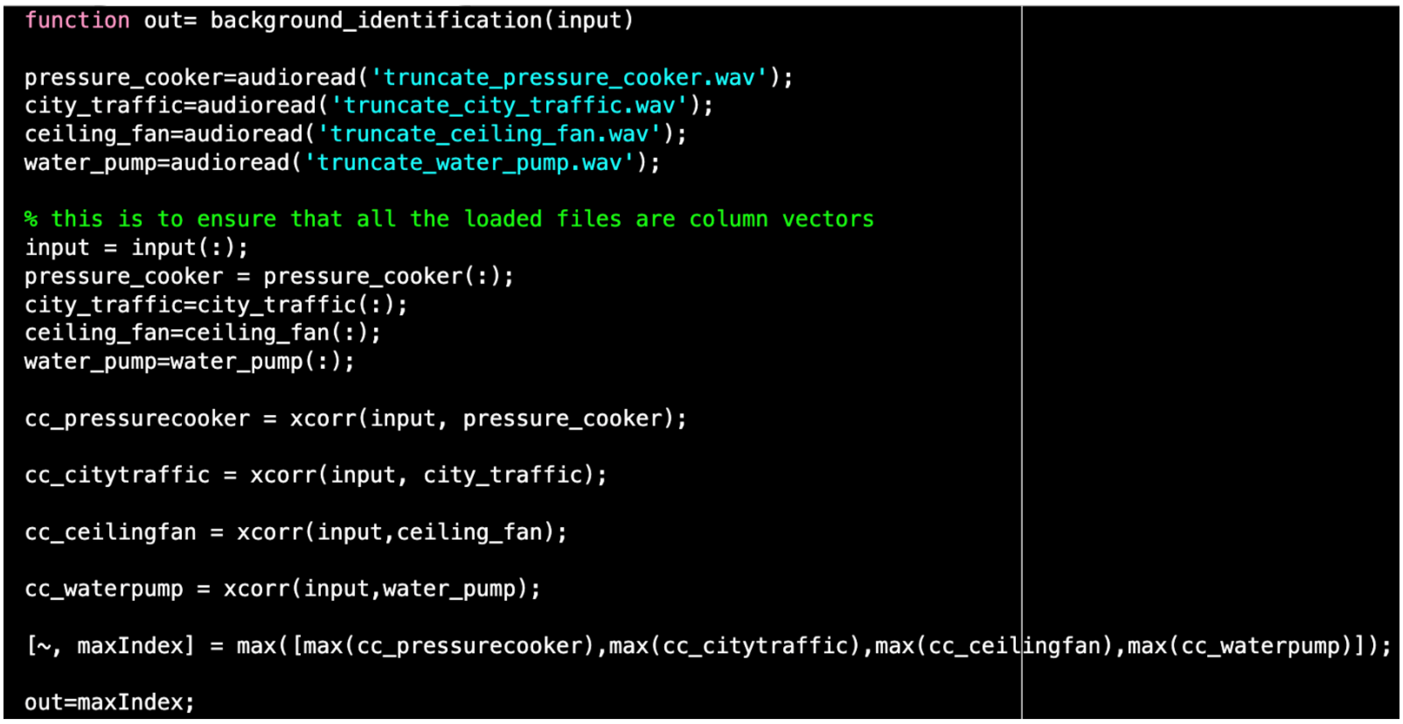






As we can see, there is a specific section of the signal unique to the each of the background noise that we have. So, we truncated these audio signals in that duration manually using the truncate_music function written and made 4 new files unique to each background noise with the names – ‘truncate_city_traffic.wav’, ‘truncate_water_pump.wav’, ‘truncate_pressure_cooker.wav’ and ‘truncate_ceiling_fan.wav’. These files are in the Output Audio folder.

The function for this question: -



```
function out= background_identification(input)

pressure_cooker=audioread('truncate_pressure_cooker.wav');
city_traffic=audioread('truncate_city_traffic.wav');
ceiling_fan=audioread('truncate_ceiling_fan.wav');
water_pump=audioread('truncate_water_pump.wav');

% this is to ensure that all the loaded files are column vectors
input = input(:);
pressure_cooker = pressure_cooker(:);
city_traffic=city_traffic(:);
ceiling_fan=ceiling_fan(:);
water_pump=water_pump(:);

cc_pressurecooker = xcorr(input, pressure_cooker);

cc_citytraffic = xcorr(input, city_traffic);

cc_ceilingfan = xcorr(input,ceiling_fan);

cc_waterpump = xcorr(input,water_pump);

[~, maxIndex] = max([max(cc_pressurecooker),max(cc_citytraffic),max(cc_ceilingfan),max(cc_waterpump)]);

out=maxIndex;
```

We take the input audio file, run cross-correlation with each of the above 4 files unique to each background noise and take the peaks in each of the outputs of cross-correlation. The peak with maximum value will correspond to the background noise since that will be the most similar to the input file.

## Cross-Correlation

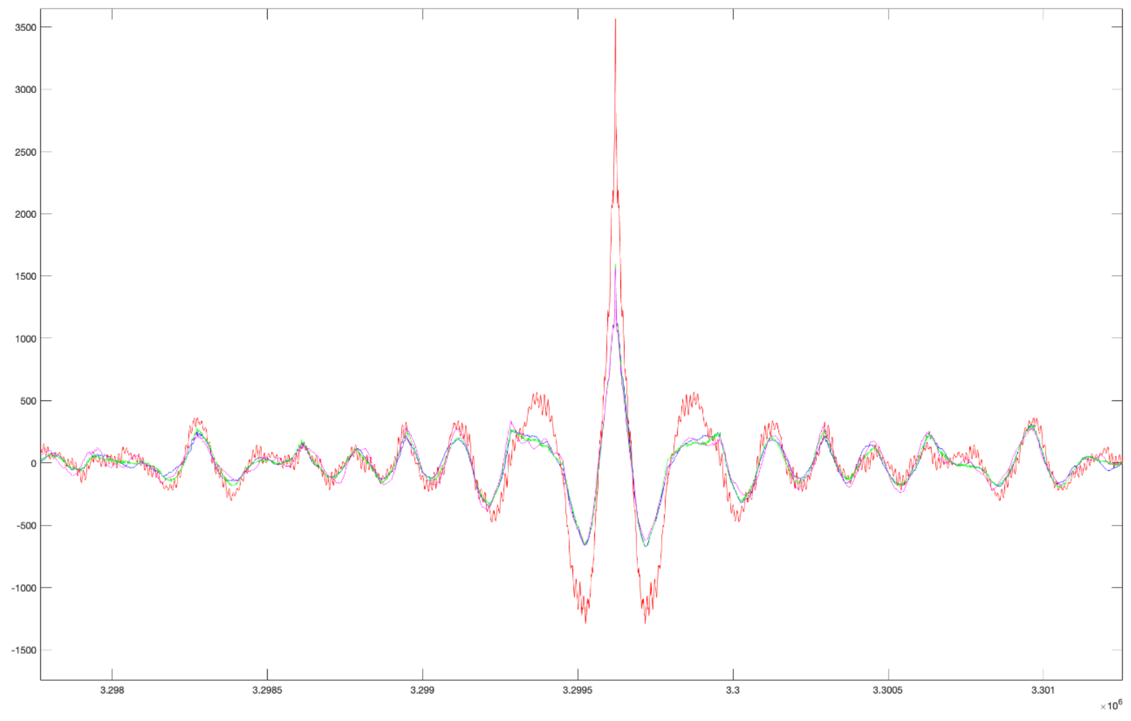
Cross correlation measures the similarity between the two signals as they vary with time. The equation for it is: -

$$(f * g)(\tau) = \int_{-inf}^{inf} f^*(t)g(t+\tau)dt$$

Here, f and g are two signals, f*(t) denotes the complex conjugate of f(t). In practical terms, the cross-correlation function essentially slides one signal over the other, multiplying corresponding values at each step and summing the results. The resulting function indicates how well the two signals match at different time lags.

## Results: -

For example, let’s take the music audio with city traffic back-ground noise and run the function and plot the 4 outputs of cross-correlation on top of each other. Shown below is the zoomed in version of the output to see it clearly.



The red color corresponds to cc_citytraffic and as we can see it has the maximum peak among all the other waveforms. Thus , we can say that the background noise corresponds to the city traffic.

## Conclusion: -

In conclusion, this method works well when we previously have some files for noise that we are trying to identify so that we can create a reference file to cross-correlate new input files with. Cross-correlation in general is a great method for pattern-recognition.