Signal Processing Lab Project - 2023
Team Noise Hunters
Team Members:

V.V.S.R Chetan Krishna
2022102047

Shaik Affan Adeeb
2022102054

G.S.S Ananya Varma
2022102064

Part 1 - Echo creation

- An echo can be simulated by adding the original signal to a delayed version of itself. This process is known as delay and sum, and it's a simple way to model the creation of echoes.

- Mathematically, if $x(n)$ represents the original signal and $x(n - n_0)$ represents a delayed version of the signal by a time delay n_0, then the signal with an added echo can be represented as:

$$y(n) = x(n) + \alpha \cdot x(n - n_0)$$

Here, α is a scaling factor that determines the amplitude of the added echo. Adjusting n_0 changes the delay time of the echo, and adjusting α changes the amplitude of the echo relative to the original signal.

Therefore:

$$y[n] = x[n] + \alpha x[n - n_0]$$

Taking Z-Transform:

$$y[z] = x[z] + \alpha x[z] z^{-n_0}$$

$$\implies \frac{y(z)}{x(z)} = 1 + \alpha z^{-n_0}$$

Transfer Function :

$$H(z) = 1 + \alpha z^{-n_0}$$

Inverse Z-Transform:

$$H(n) = \delta[n] + \alpha \cdot \delta[n - n_0]$$

- After obtaining the transfer function $H(z) = 1+\alpha z^{-n_0}$, $y[n]$ can indeed be obtained by convolving $x[n]$ with the impulse response $h[n]$.

$$\boxed{y[n] = x[n] * h[n]}$$

$$\implies y[n] = x[n] * (\delta[n] + \alpha \cdot \delta[n - n_0])$$

$$\implies y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Expanding the expression, we obtain:

$$\boxed{y[n] = x[n] + \alpha \cdot x[n - n_0]}$$

The result reaffirms that the output $y[n]$ is indeed the sum of the original signal $x[n]$ and a scaled and delayed version of itself, which aligns with the original equation $y[n] = x[n] + \alpha x[n - n_0]$.

- Also the transfer function $H(z) = 1 + \alpha z^{-n_0}$, represents a comb filter in the Z-domain.Directly this comb filter can be applied to the input signal $x[n]$ to produce the output signal $y[n]$.

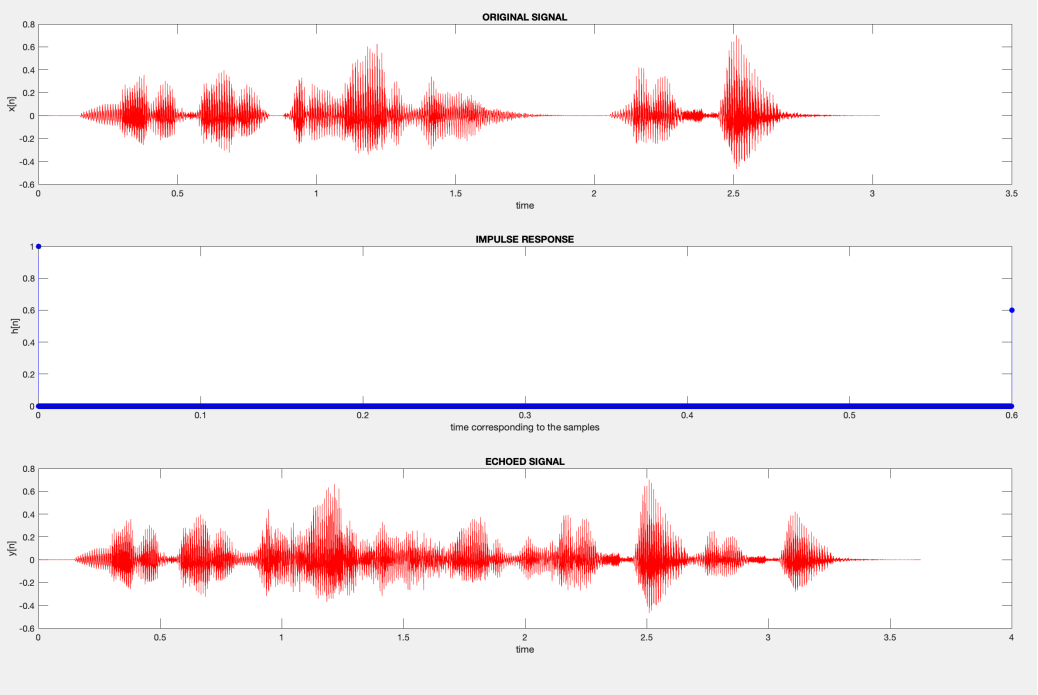
Matlab Code :

```
%% original signal is x[n].
%% taking the echoed signal to be y[n]
%% since echoed signal is the sum of the orignal and the delayed signal:::
%% echoed signal is y[n] = x[n] + alpha * x[n-N0]
%% alpha : attenuation parameter (0,1) , and N0 : no. of samples in delayed time period

%% finding the signal from the wav file
[x, Fs] = audioread("q2_easy.wav");
d = 30;
delay = 2 * d / 330;
N0 = ceil(delay * Fs);
% hence the impulse response of the system is h[n] = del[n] + alpha * del[n-N0]
h = zeros(1,N0+1);
time_index_echoed_signal = 0 : 1/Fs : (length(h)-1)/Fs;
% giving sample weights to the impulse response
alpha = 0.6;
h(1) = 1;
h(N0+1) = alpha;

y = conv(x(:,1),h,"full");
time_of_y =  length(y)/Fs;
```

```
% disp(time_of_y);
% disp(length(y));
% display(time_of_y);
 attack = 1000;
 decay = 100000;
 release = 400000;
 sustain = 0.7;
 sustain_duration = length(y)-(attack+decay+release);

env = envelope(attack,decay,0.7,sustain_duration,release
   );
disp(length(env));
disp(length(y));
time_convoluted_signal = 0 : 1/Fs : (length(y)-1)/Fs;
 % final_echoed_signal = y .*env;

% for k = 1 : length(y)
%     y(k) = y(k) * env(k);
% end

figure;
subplot(3,1,1);
plot((0 : 1 : length(x(:, 1))-1)/Fs, x, 'Color','r');
xlabel("time");
ylabel("x[n]");
title("ORIGINAL SIGNAL");
xlim([0,10000/Fs]);
sound(x,Fs);
pause((length(x)/Fs));

subplot(3,1,2);
stem(time_index_echoed_signal, h, "filled",'Color','b');
xlabel("time corresponding to the samples");
ylabel("h[n]");
title("IMPULSE RESPONSE");

subplot(3,1,3);
plot(time_convoluted_signal,y,'Color','r');
xlabel("time");
ylabel("y[n]");
title("ECHOED SIGNAL");
xlim([0,10000/Fs]);

```

```
64 sound(y,Fs);
65
66
67 function env = envelope(a,d,s,sd,r)
68 t_total = a+d+sd+r;
69 sample_indices = 0 : 1 : t_total-1;
70 % t_env = 0 : 1/fs : t_total-1/fs;
71 % t_env = [t_env, a+d+sd+r];
72 env = zeros(size(sample_indices));
73  % attack
74  for k = 1 : length(sample_indices)
75      if(sample_indices(k)<=a)
76          env(k) = sample_indices(k)/a;
77      end
78
79      if(sample_indices(k)>=a && sample_indices(k)<=a+d)
80          env(k) = 1 + ((s-1)/d)* (sample_indices(k) - a)
                ;
81      end
82
83      if(sample_indices(k)>=a+d && sample_indices(k)<=a+d
           +sd)
84          env(k) = s;
85      end
86
87      if(sample_indices(k)>=a+d+sd  &&  sample_indices(k)
           <=a+d+sd+r)
88          env(k) = -1 * (s/r) * (sample_indices(k) - a-d-
                sd-r);
89      end
90  end
91 end
```

Listing 1: Part 1 - Echo creation


ORIGINAL SIGNAL
x[n]
time
IMPULSE RESPONSE
h[n]
time corresponding to the samples
ECHOED SIGNAL
y[n]
time

Q1 hindi

Part 2 - Cancel the echo

We have done this in two ways, below is approach 1 :

Adaptive filters :

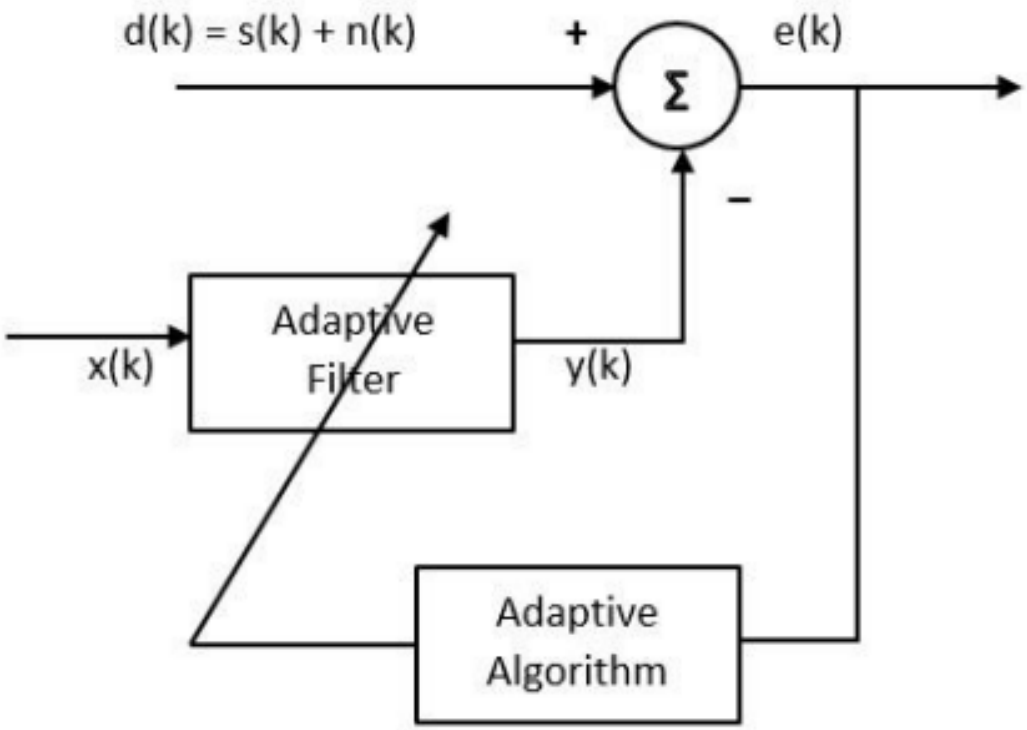


Figure 1: General setup of adaptive filter

- The general set up of an adaptive filtering environment is shown above, where k is the iteration number, x(k) denotes the input signal, y(k) is the adaptive filter output, and d(k) defines the desired signal. The error signal e(k) is calculated as d(k)-y(k). The error is then used to form a performance function or objective function that is required by the adaptive algorithm in order to determine the appropriate updating of the filter coefficients

An adaptive filter adapts techniques that must be implemented to obtain accurate solutions to this problem of echo in an echo cancellation system. An adaptive filtering process involves:

1. A filtering process with which a desired output in response to an input data is designed to produce, and
2. An adaptive process that provides an algorithm for modifying a set of filter coefficients.

- Adaptive filters generally consist of two distinct parts as in:
  A filter whose structure is designed to perform the desired processing function, and an adaptive algorithm for adjusting the parameters (coefficients) of that filter . The major challenge of an echo canceller is in choosing a suitable adaptive algorithm to adjust the coefficients of the adaptive filter in order to match the desired response

In choosing a suitable algorithm, their performances has to be considered in order to get the desired result, some important performance measures include convergence rate, mean square error, computational complexity, stability, and filter length.

1. **Convergence rate:** Adaptive filters attempt to minimize the power of the error signal by iteratively adjusting the filter coefficients. **The process of minimizing the power of the error signal is known as convergence**. **A fast convergence indicates that the adaptive filter takes a short time to calculate the appropriate filter coefficients that minimize the power of the error signal**. Convergence rate is how fast or slow a filter converges to its resultant state, in most cases, the faster the better. However, the convergence rate of an adaptive filter is not independent of other performance measures, there is usually a trade off with other performance criteria

2. **Mean Square Error (MSE):** This is how much a system can adapt to a given solutions, a small MSE is an indication that the adaptive system has accurately modeled, predicted, adapted, and/or converged to a solution for the system, there are number of factors which can help determine the MSE including, but not limited to; quantization noise, order of the adaptive systems, measurement noise, and error of the gradient due to the finite step size.

3. **Computational complexity:** This is important in real time adaptive filter applications. When a real time system is being implemented, there are hardware limitations that may affect the performance of the system. A high complex algorithm will require much greater hardware resources than a simplistic algorithm .

4. **Stability:** The stability of an adaptive filter is probably the most important performance measure as the power of the error signal converges to zero. In some cases, the power of the error signal does not converge to zero, but instead,diverges, this shows that the adaptive filter is unstable. The stability of an adaptive filter can be determined by the step size of the input.

5. **Filter Length:** The filter length is inherently tied up to many of the other performance measures. It specifies how accurately a given system can be modeled by the adaptive filter. In addition the filter length affects the convergence rate, by increasing or decreasing computational time, it can affect the stability of the system, at certain step sizes, and it affects the MSE. If the filter length of the system is increased, the number of computations will increase, decreasing the maximum convergence rate .

6. **Signal to Noise Ratio:** Noise reduction ratio (Nrr) is the ratio of the noise power to error power. It measures the signal quality of recovered signal. In this case, echo is noise.

- Least Mean Square (LMS) algorithm, Normalized Least Mean Square (NLMS) algorithm (which is a modified form of the standard LMS algorithm), and Recursive Least Squares (RLS) algorithm. These three methods are employed for echo removal.

Normalized least mean square (NLMS) algorithm :
We have used this as it is the most suitable algorithm for an echo cancellation due to its faster convergence rate compared to other algorithms with a very little extra computational complexity than the LMS algorithm. This algorithm updates the coefficients by using the following procedure:

1. Filter Output:
$$y(k) = \hat{w}^H(k)x(k)$$
2. Estimated Error:
$$e(k) = d(k) - y(k)$$
3. Tap-weight Adaptation:
$$\hat{w}(k+1) = \hat{w}(k) + \frac{\mu e^*(k)x(k)}{\varepsilon + x^H(k)x(k)}$$

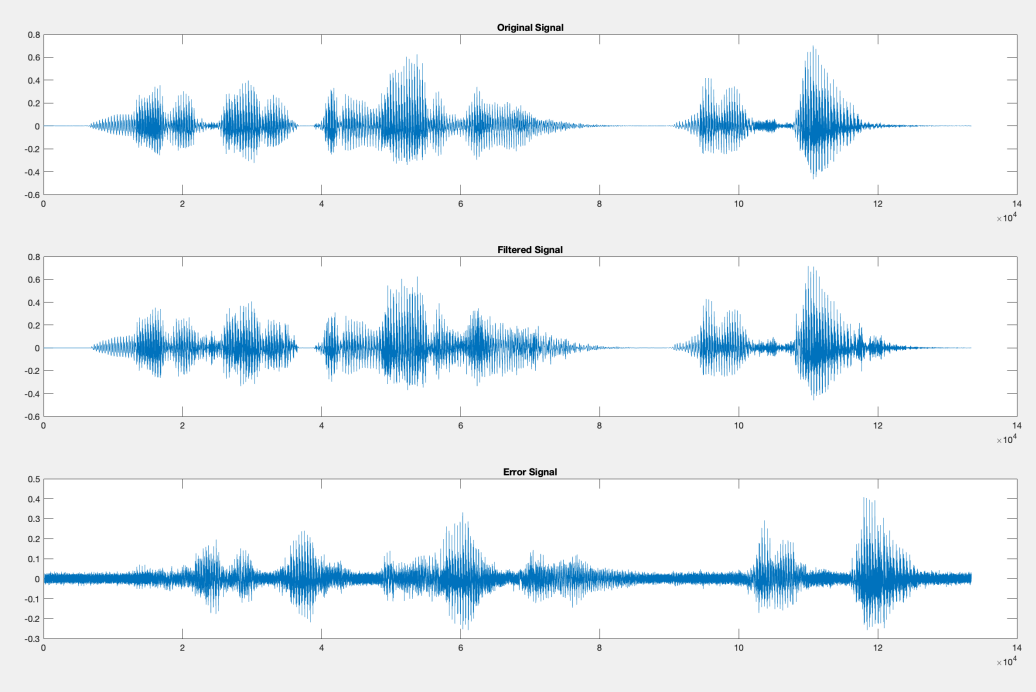
Where $\hat{w}(k)$ represents the filter coefficients vector, $x(k)$ is the tap input vector, $d(k)$ is the desired filter output, $y(k)$ is the output of the adaptive filter, $e(k)$ is the error, μ is a scaling factor called step-size, and ε is the regularization parameter to avoid dividing by zero.

Implementation of NLMS algorithm in matlab :

```
[x, fs] = audioread("hindi_2s.wav");
[Xe, fs1] = audioread("hindi.wav");

% sound(x, fs);
% pause(length(x)/fs + 0.1);

[filtered_signal, error_signal] = nlms(x, Xe, 128, fs1);

sound(filtered_signal, fs1);
% pause(length(filtered_signal)/fs1 + 0.1);

figure;
subplot(3,1,1);
plot(x);
title('Original Signal');

subplot(3,1,2);
plot(filtered_signal);
title('Filtered Signal');

subplot(3,1,3);
plot(error_signal);
title('Error Signal');

function [filtered_signal, error_signal] = nlms(x, di,
   sysorder, fs)
    N = length(x);
    b = fir1(sysorder - 1, 0.5);
    e = filter(b, 1, di);
    n = 0.01 * randn(length(e), 1);
    d = e + n;
    w = zeros(sysorder, 1);

    [filtered_signal, error_signal] = nlmsAlgorithm(x, d
       , w, sysorder, N);
end

function [filtered_signal, error_signal] = nlmsAlgorithm
   (x, d, w, sysorder, N)
    filtered_signal = zeros(N, 1);
    error_signal = zeros(N, 1);
```

```
39
40      for n = sysorder:N
41          u = x(n:-1:n - sysorder + 1);
42          y = w' * u;
43          error_signal(n) = d(n) - y;
44          mu = 0.5 / (1 + u' * u);
45          w = w + mu * u * error_signal(n);
46          filtered_signal(n) = y;
47      end
48  end
```

Listing 2: Adaptive filters approach



Q2 hindi (nlms)

Another approach using Autocorrelation method:

We tried to cancel the echo using the autocorrelation method by coming up with a logic :

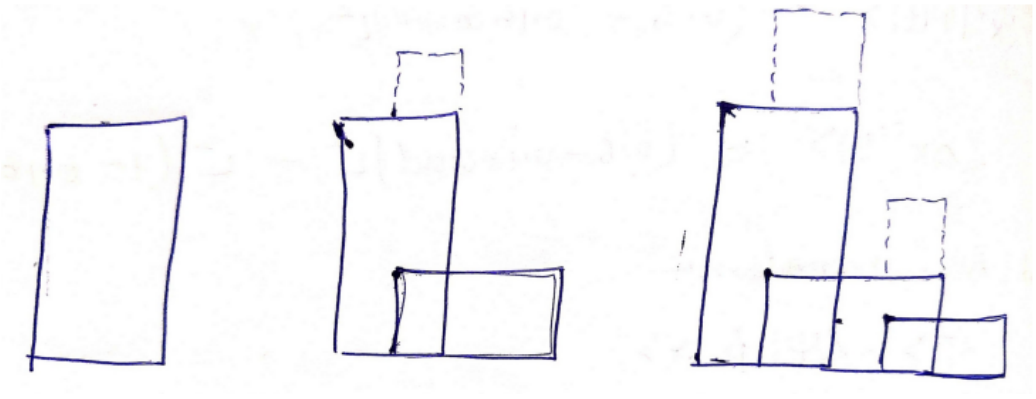
1. Autocorrelate the echoed signal with itself using the `xcorr` function:

$$[y, \text{lag}] = \text{xcorr}(\text{echo\_signal}, \text{echo\_signal}, \text{Fs});$$

This function samples the signal with the sampling frequency of Fs and correlates the given two signals.

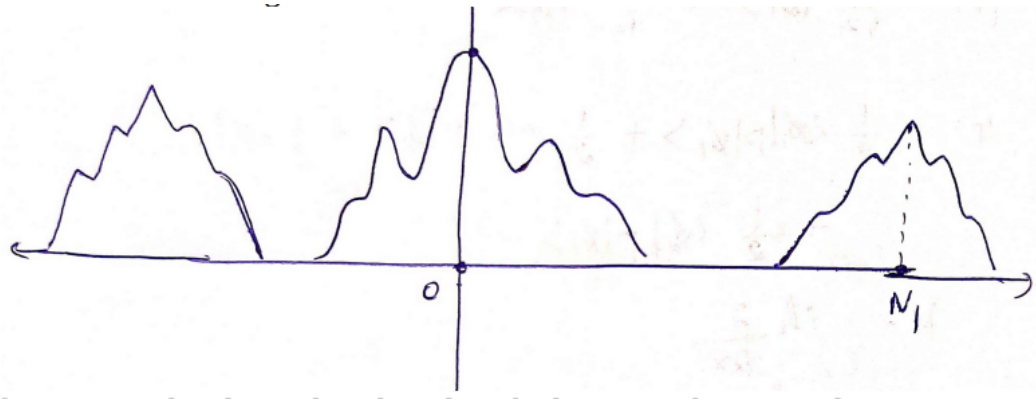
2. Correlation provides the correspondence of one signal with the other. Therefore, the time instances where the signal overlaps with itself or its attenuated part can be observed as peaks in the autocorrelated signal.

   For example, when we consider the signal given in the 2nd question `q2_not_so_easy.wav`, some part of the mixed echoed signal...



Part of mixed echoed signal

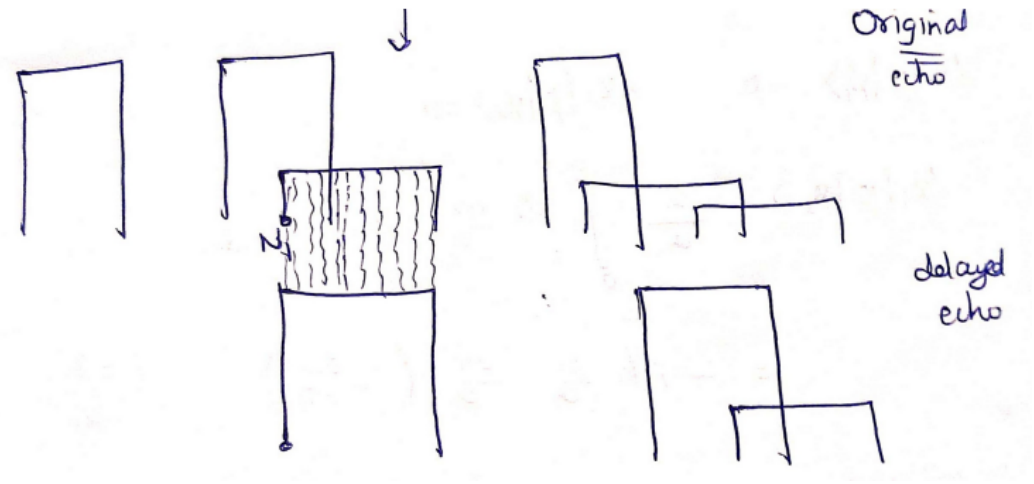
The autocorrelated signal looks like :



Auto correlated signal

The autocorrelated signal is plotted on the lag axis. The time index $N1$ corresponds to the 1st peak of the autocorrelated signal when it coincides with its first attenuated version. The $N1$ point is where the 1st echoed attenuated part of the original signal overlaps with the original signal.

Now, we delay the autocorrelated signal up to $N \times \text{Fs}$ samples to correlate itself with its attenuated part.



After delaying the echoed signal, we divide the corresponding samples. In this example, the first set of outputs would be greater than 1, and the second set of outputs would be less than 1, providing us with the attenuation of the first echo. This process involves delaying the echoed part every time there is a peak in the correlated signal and dividing the corresponding samples to find the attenuation of that echoed part.

To find the time instances of the peaks, we used the `islocalmax` function in MATLAB.

Algorithm:

1. Find the first set of local maxima. The `islocalmax` function maps the local maxima to 1's and non-local maxima samples to 0's. When we apply `islocalmax` again, we would get the same set of samples, as the adjacent samples to the previously calculated local maxima are 0's. So, every time we find the local maxima, we remove the zeros (map zeros part to a null array) and join the samples to get a new set of local maxima when we run the function again. We record the time instances where local maxima are achieved because those delays are used to delay the echoed signal.

2. We recursively follow this process until one step before we get the array size as 1 (i.e., before we get the global maxima). One step before we get the global maxima, we obtain all set peaks (main local maxima of interest).

This is the process we implemented for the 2nd question using the auto-correlation method.

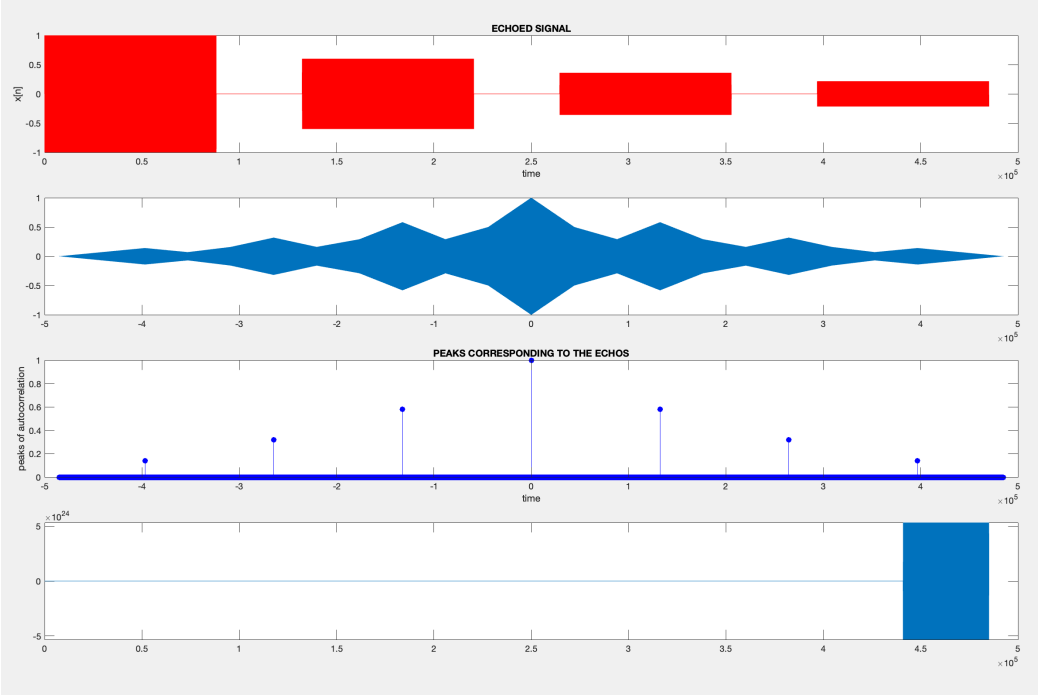
Matlab Implementation :

```
%% echoed signal
[x_echoed, Fs]= audioread("q2_not_so_easy.wav");
figure;

subplot(4,1,1);
plot(x_echoed,'Color','r');
xlabel('time');
ylabel("x[n]");
title("ECHOED SIGNAL");
sound(x_echoed,Fs);
% pause(length(x_echoed)/Fs);

[y,lag] = xcorr(x_echoed(:,1),x_echoed(:,1),'normalized');
y_temp = y;
y_correlated= y;
y_temp_for_delays_indicex = zeros(1, length(y_temp));
subplot(4,1,2);
plot(lag,y);
length_local_max = 2;
while(length_local_max > 1)
    y_previous = y;
    local_max_points = islocalmax(y);
    y = y.*local_max_points;
    local_max_points(local_max_points==0) = [];
    y(y==0) = [];
    length_local_max = length(y);
end
local_max_indices_final = zeros(1, length(y_previous));
for k = 1 : length(y_previous)
    samples_check_for_index = y_previous(k);
    for m = 1 : length(y_temp)
        if(y_temp(m) == samples_check_for_index)
            y_temp_for_delays_indicex(m) =
               samples_check_for_index;
            y_temp(m)=0;
            local_max_indices_final(k) = m;
            break;
```

```
        end

    end
end

subplot(4,1,3);
stem(lag,y_temp_for_delays_indicex,"filled",'Color','b');
xlabel('time');
ylabel('peaks of autocorrelation');
title("PEAKS CORRESPONDING TO THE ECHOS");


%% delaying the correlated sigal and subtracting the echos
%%
% after delaying the correlated signal we divide the corresponding samples
% the minimum of the divided values will be the attenuation factor for the
% corresponding echo

attenuations = zeros(1, length(local_max_indices_final));
for k = 1 : length(local_max_indices_final)
    delay = local_max_indices_final(k);
    component_to_be_delayed = zeros(1, delay);
    for m = 1 : delay
        component_to_be_delayed(m) = y_correlated(m);
    end
    delayed_signal = zeros(1, 2*delay-1);
    for l = delay : length(delayed_signal)
        delayed_signal(l) = component_to_be_delayed(l-delay+1);
    end

    Corresponding_division_array = zeros(1, delay);
    for n = 1 : delay
        if((n+delay-1) <= length(y_correlated))
        Corresponding_division_array(n) = y_correlated(n+delay-1)/delayed_signal(n+delay-1);
        if(Corresponding_division_array(n)<1)
```

```
72              attenuations(k) =
                    Corresponding_division_array(n);
73              break;
74          end
75          end
76      end
77      % if(min(Corresponding_division_array)<=1)
78          % attenuations(k) = min(
                Corresponding_division_array);
79      % end
80  end
81  disp(attenuations);
82  %% impulse response
83  % numerator
84  % h = zeros(1, length(x_echoed));
85  num = zeros(1, length(x_echoed));
86  num(1) = 1;
87  for k = 1 : length(local_max_indices_final)
88      delay = local_max_indices_final(k);
89      num(local_max_indices_final(k))=delay;
90  end
91  den = 1;
92  audio_without_echo = filter(den,num,x_echoed);
93  subplot(4,1,4);
94  plot(audio_without_echo);
95  sound(audio_without_echo);
```

Listing 3: Autocorrelation approach

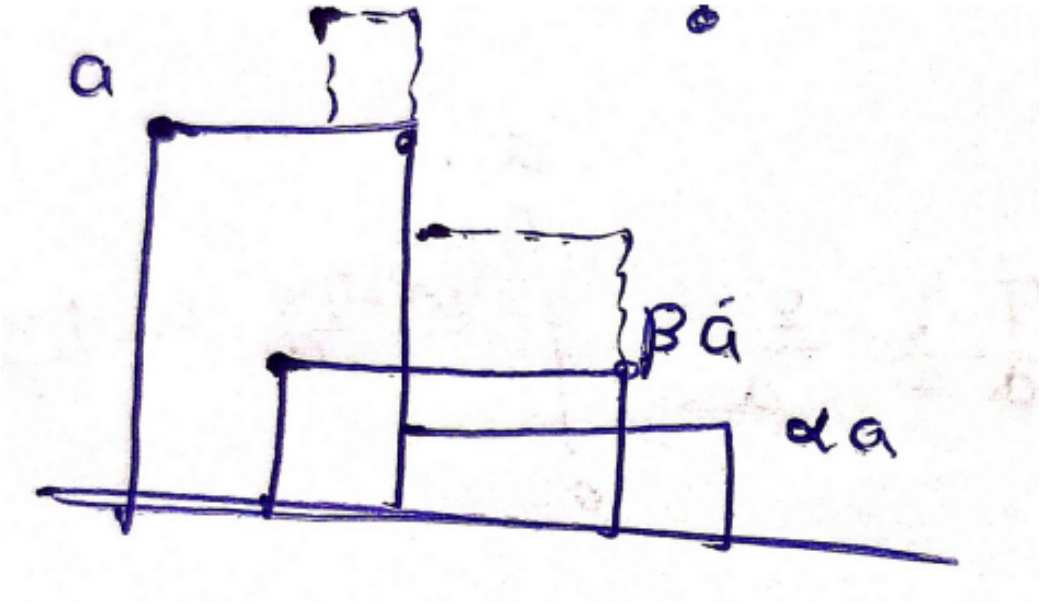


Autocorrelation approach for q2 easy signal

However, there are some limitations to this approach, and it will work for only particular echoed signals.

1. Dividing the samples indicates dividing the samples of the sinusoidal signals. However, when we delay the signal, all the samples may not achieve the same deviation, and the phase change may not be the same, which may not lead to the division of the corresponding samples.

2. This approach works when there are at most 2 overlaps at any instance. For example ,(which is shown in figure below)



When the echoed signal is delayed to correspond with the 2nd attenuated version of the signal, all the divided outputs of the samples do not directly give the attenuation because the 3rd attenuated signal overlaps completely with the 2nd signal.

Part 3: What is this noise?

In this section, we will work on the task of classifying background noise in music recordings without removing the noise. Our objective is to accurately identify and categorize the type of noise present in the recording, distinguishing the source of origin.

The types of noises can be:

- Fan
- Pressure cooker
- Water Pump
- Traffic

We tried to implement the 3rd question in two ways:

1st Approach: Correlating the signal with the reference signal

- Correlate a music signal containing noise (e.g., pressure cooker) with another music signal containing noise. If the noise component in both correlated signals is the same, there would be a peak corresponding to the overlap.
- Peaks in the correlation correspond to the same noise, and so we can direclty predict the noise present in the background.

2nd Approach: Short-Time Fourier Transform (STFT)

- The Short-Time Fourier Transform (STFT) is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.
- The procedure for computing STFTs involves dividing a longer time signal into shorter segments of equal length and then computing the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment, and one can plot the changing spectra as a function of time.
- The window size should be as long as possible to consider as many frequency components as possible. We used a Hann window of size 4096 with an overlap of 2048 length.

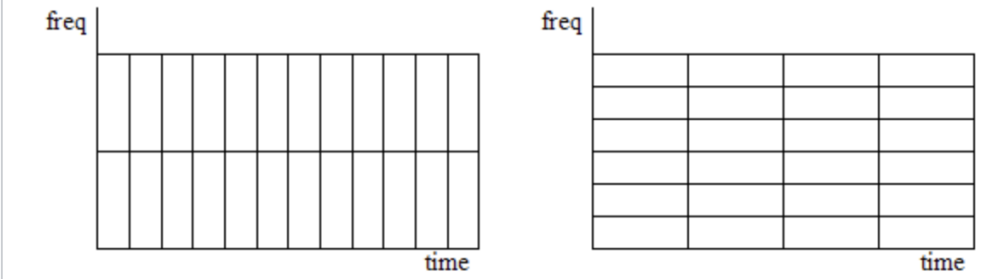
- The overlap should be as high as possible but less than the window length to avoid attenuationn because larger overlap length ensures that we are calculate the strength of particular frequency at almost all the time instances which ensures effective fourier transform of the signal.

Process

1. Find the frequencies of the noise signals (take sample frequencies close to those noise signals).

2. Take the STFT of the given music signal.

3. The STFT results in a multidimensional array of dimension $m \times n$, where m is frequencies (window length) and n is time instances. This



Comparison of STFT resolution. Left has better time resolution, and right has better frequency resolution.

is one of the reasons for the creation of the wavelet transformation and multiresolution analysis, which can provide good time resolution for high-frequency events and good frequency resolution for low-frequency events—the combination best suited for many real signals.

From that multidimensional $m \times n$ array obtained from the STFT, we find the noise frequency or the frequency close to that. We then take the root mean squared (RMS) value of the strengths of that frequency component in the signal.

4. For the given four audio samples, we get the strength of each component of noise (ceiling fan, pressure cooker, etc.) in the given audio signal.

5. Now, the maximum of all the strengths is taken, and the noise that corresponds to the maximum strength is classified as the noise present in the background.

Matlab Implementation with stft approach:

```
function main()
    inputFile = "music_water-pump_hp.wav";
    noiseType = classifyNoise(inputFile);
    disp('The detected noise type is: ' + noiseType);

    [ampCeilingFan, fCeilingFan] = processAudioFile("
       music_ceiling-fan_hp.wav");
    plotSignal(fCeilingFan, ampCeilingFan, "Ceiling Fan
       ");

    [ampTraffic, fTraffic] = processAudioFile("
       music_city-traffic_hp.wav");
    plotSignal(fTraffic, ampTraffic, "Traffic");

    [ampPressureCooker, fPressureCooker] =
       processAudioFile("music_pressure-cooker_hp.wav");
    plotSignal(fPressureCooker, ampPressureCooker, "
       Pressure Cooker");

    [ampWaterPump, fWaterPump] = processAudioFile("
       music_water-pump_hp.wav");
    plotSignal(fWaterPump, ampWaterPump, "Water Pump");

    [ampInput, fInput] = processAudioFile(inputFile);
    plotSignal(fInput, ampInput, "Input");
end

function plotSignal(frequency, amplitude, titleText)
    figure;
    plot(frequency, amplitude);
    title(titleText);
end

function [amp, frequency] = processAudioFile(fileName)
    [y, fs, ~, ~] = loadAudioFile(fileName);
    [s, f, ~] = computeSTFT(y, fs);
    amp = analyzeSignal(s);
    frequency = f;
end

function [y, fs, L, T] = loadAudioFile(inputFile)
```

```
    [y, fs] = audioread(inputFile);
    L = length(y);
    T = L / fs;
end

function [s, f, t] = computeSTFT(y, fs)
    [s, f, t] = stft(y(:, 1), fs, 'Window', hann(4096),
        'OverlapLength', 1024, 'FFTLength', 4096);
end

function amp = analyzeSignal(s)
    S = sum(transpose(abs(s)));
    amp = S / max(S);
end

function noiseType = classifyNoise(inputFile)
    [ampCeilingFan, ~] = processAudioFile("music_ceiling
        -fan_hp.wav");
    [ampTraffic, ~] = processAudioFile("music_city-
        traffic_hp.wav");
    [ampPressureCooker, ~] = processAudioFile("
        music_pressure-cooker_hp.wav");
    [ampWaterPump, ~] = processAudioFile("music_water-
        pump_hp.wav");
    [ampInput, ~] = processAudioFile(inputFile);

    rCeilingFan = rmse(ampCeilingFan, ampInput);
    rTraffic = rmse(ampTraffic, ampInput);
    rPressureCooker = rmse(ampPressureCooker, ampInput);
    rWaterPump = rmse(ampWaterPump, ampInput);

    rArray = [rCeilingFan, rTraffic, rPressureCooker,
        rWaterPump];
    r = min(rArray);

    if r == rmse(ampCeilingFan, ampInput)
        noiseType = "Ceiling Fan";
    elseif r == rmse(ampTraffic, ampInput)
        noiseType = "Traffic";
    elseif r == rmse(ampPressureCooker, ampInput)
        noiseType = "Pressure Cooker";
    elseif r == rmse(ampWaterPump, ampInput)
        noiseType = "Water Pump";
```

```
    end
end
```

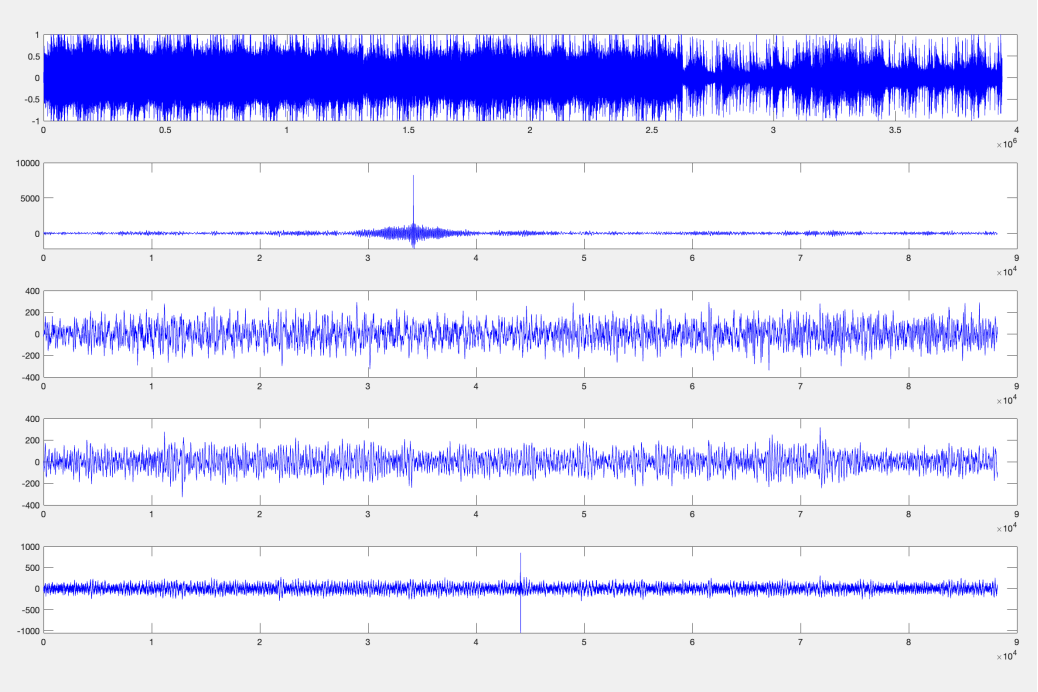
Listing 4: STFT approach

Matlab implementation for with correlation approach :

```
%% FFT's of the given noise files
[music_water_pump,Fs_water] = audioread("music_water-pump.wav");
[music_pressure_cooker, Fs_cooker] = audioread("music_pressure-cooker.wav");
[music_city_traffic,Fs_traffic] = audioread("music_city-traffic.wav");
[music_ceiling_fan,Fs_fan] = audioread("music_ceiling-fan.wav");

[music_ceiling_fan_test,Fs_fan_test] = audioread("music_ceiling-fan_hp.wav");
[music_city_traffic_test, Fs_traffic_test] = audioread("music_city-traffic_hp.wav");
[music_pressure_cooker_test, Fs_cooker_test] = audioread("music_pressure-cooker_hp.wav");
[music_water_pump_test, Fs_water_test] = audioread("music_water-pump_hp.wav");

figure;
mixed_signal = [music_pressure_cooker_test; music_ceiling_fan_test;music_city_traffic_test];
subplot(5,1,1);
plot(mixed_signal,'Color','b');

[mixed_resenmble_ceiling_fan,lag1] = xcorr(mixed_signal, music_ceiling_fan,Fs_fan);
subplot(5,1,2);
plot(mixed_resenmble_ceiling_fan,'Color','b');


[mixed_resenmble_water_pump, lag2] = xcorr(mixed_signal, music_water_pump,Fs_traffic);
subplot(5,1,3);
plot(mixed_resenmble_water_pump,'Color','b');

```

```
[mixed_resenmble_traffic,lag3] = xcorr(mixed_signal,
    music_city_traffic,Fs_cooker);
subplot(5,1,4);
plot(mixed_resenmble_traffic,'Color','b');

[mixed_resenmble_cooker,lag4] = xcorr(mixed_signal,
    music_pressure_cooker,Fs_water);
subplot(5,1,5);
plot(mixed_resenmble_cooker,'Color','b');
```

Listing 5: Correlation approach



Correlation approach