B a)

(i)

From part A question 2, the impulse response of

$$H_1(z) = 1 + \alpha z^{-1} = \alpha^0 z^{-0} + \alpha z^{-1} \tag{1}$$

can be shown as

$$h_1[n] = \sum_{i=0}^{1} \alpha^i \delta[n-i] = \alpha^0 \delta[n] + \alpha\delta[n-1] \tag{2}$$

Now the filter becomes

$$H_1(z^D) = 1 + \alpha z^{-D} = \alpha^0 z^{-0D} + \alpha z^{-D} \tag{3}$$

Using time shift property, the corresponding impulse response is

$$h_1'[n] = \alpha^0 \delta[n] + \alpha\delta[n-D] = \delta[n] + \alpha\delta[n-D] \tag{4}$$

If we have an input signal $x[n]$ pass through the filter $H(z^D)$, the output $y[n]$ is

$$y[n] = x[n] + \alpha x[n-D] \tag{5}$$

It can be clearly seen that output signal $y[n]$ is the superposition of the original signal: $x[n]$ and the delayed and attenuated signal: $\alpha x[n-D]$, i.e. "echo".

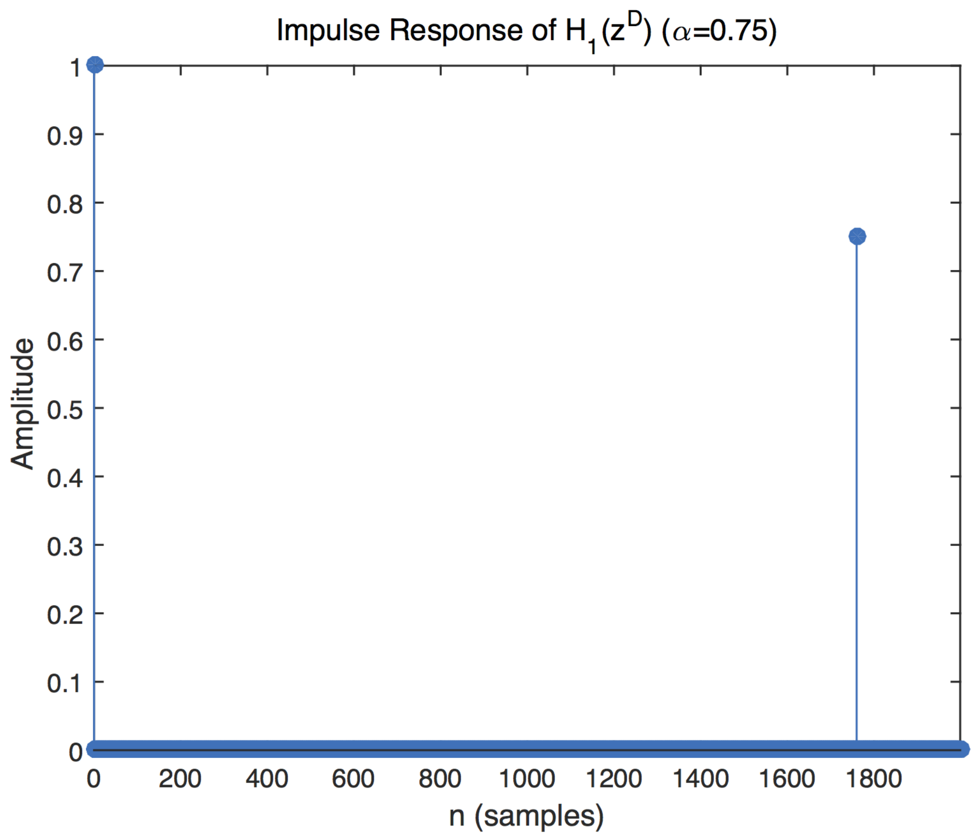

Figure 1: Impulse Response $H_1(z^D)$ ($\alpha = 0.75$)

(ii)

Here we set the sampling rate $F_s = 8$ kHz.

$$y[n] = x[n] + \alpha x[n-D]$$
$$y(t) = x(t) + \alpha x(t - \frac{D}{F_s})$$

$\frac{D}{F_s} = 220$ ms $\longrightarrow D = 220$ ms $\times$ 8 kHz $= 1760$ samples

α determines the amplification of the original signal. Due to $0 < \alpha < 1$, the echo is essentially attenuated. For instance, when $\alpha = 0.1$, the echo amplitude is only 10% of the magnitude of the original signal.

Larger D implies more delay (samples), in other words, longer time delay(ms).

(iii)

The c code can be seen in Appendix.

(iv)

We can hear a sound followed by its echoes every time we play it.

B b)

(i)

The frequency response of filter $H_2(z^D)$ is

$$\begin{aligned} H_2(e^{j\omega D}) &= \frac{1}{1+\alpha e^{-j\omega D}} \\ &= \sum_{n=0}^{\infty}(-\alpha)^n e^{-jnD\omega} \end{aligned}$$

Referring to the conclusion in part A, the impulse response of filter $H_2(z^D)$ is

$$h_2[k] = \sum_{k=0}^{\infty}(-\alpha)^k \delta[n-kD] \tag{6}$$

$$h[n] = \begin{cases} (-\alpha)^n & n = kD, k \in \mathbb{Z}_0^+ \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Response to an arbitrary sequence $\{x[n]\}$ is shown below

$$
\begin{aligned}
x[n] &= \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \\
y[n] &= \sum_{k=-\infty}^{\infty} x[k]h[n-k] \\
&= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \\
&= \sum_{k=0}^{\infty}(-\alpha)^k x[n-kD] \quad (|\alpha|<1)
\end{aligned}
$$

It can be clearly seen that output signal $y[n]$ is the superposition of the original signal: $x[n]$ and an infinite sequence of delayed and attenuated signals: $(-\alpha)^k x[n-kD]$. Hence, multiple echoes are generated.

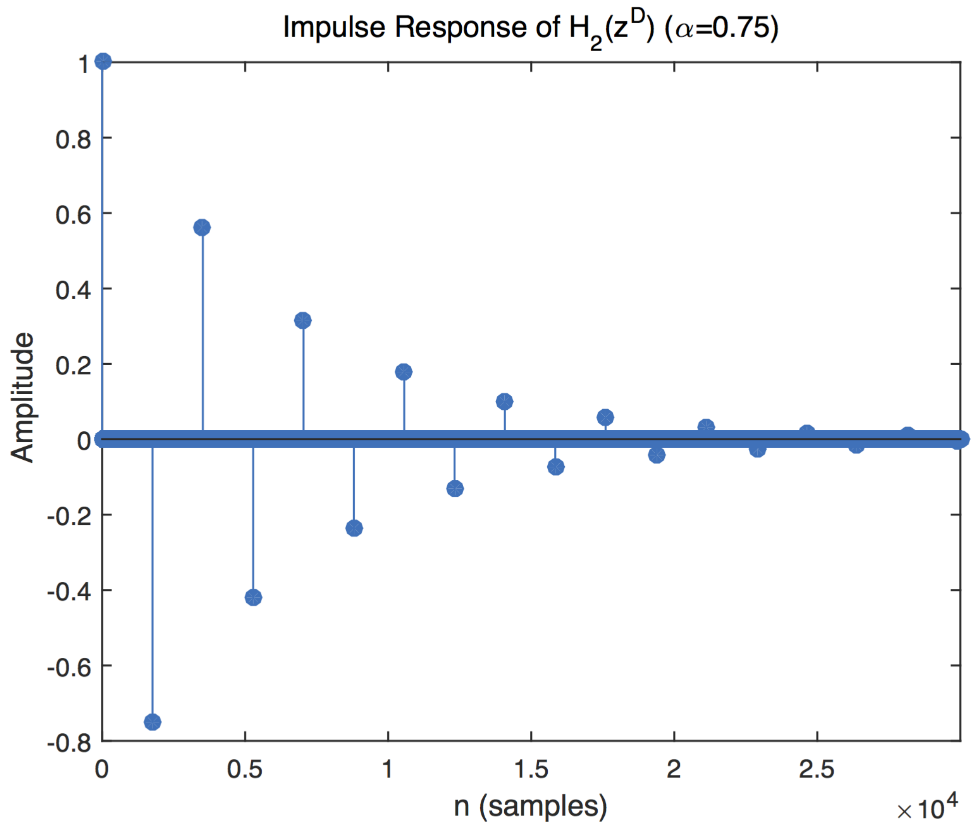

Figure 2: Impulse Response $H_2(z^D)$ $(\alpha = 0.75)$

(ii)

$$
\begin{aligned}
y[n] &= x[n] + (-\alpha)x[n-D] + (-\alpha)^2 x[n-2D] + (-\alpha)^3 x[n-3D] + \cdots \\
y(t) &= x(t) + (-\alpha)x(t-\frac{D}{F_s}) + (-\alpha)^2 x(t-\frac{2D}{F_s}) + (-\alpha)^3 x(t-\frac{3D}{F_s}) + \cdots
\end{aligned}
$$

$\frac{D}{F_s} = 220\text{ ms} \longrightarrow D = 220\text{ ms} \times 8\text{ kHz} = 1760\text{ samples}$

More echoes and more obvious delays can be heard from the signal generated in b) (ii). The signal generated in b) (ii) sounds louder.

(iii)

$$
\begin{aligned}
\sum_{k=0}^{\infty} h_1^2[k] &= \sum_{k=0}^{\infty} h_2^2[k] \\
1^2 + \alpha_1^2 &= 1^2 + (-\alpha_2)^2 + (\alpha_2^2)^2 + (-\alpha_2^3)^2 + \cdots \\
1 + \alpha_1^2 &= \frac{1}{1-\alpha_2^2} \\
\alpha_2 &= \sqrt{1 - \frac{1}{1+\alpha_1^2}} \\
\alpha_2 &= 0.6
\end{aligned}
$$

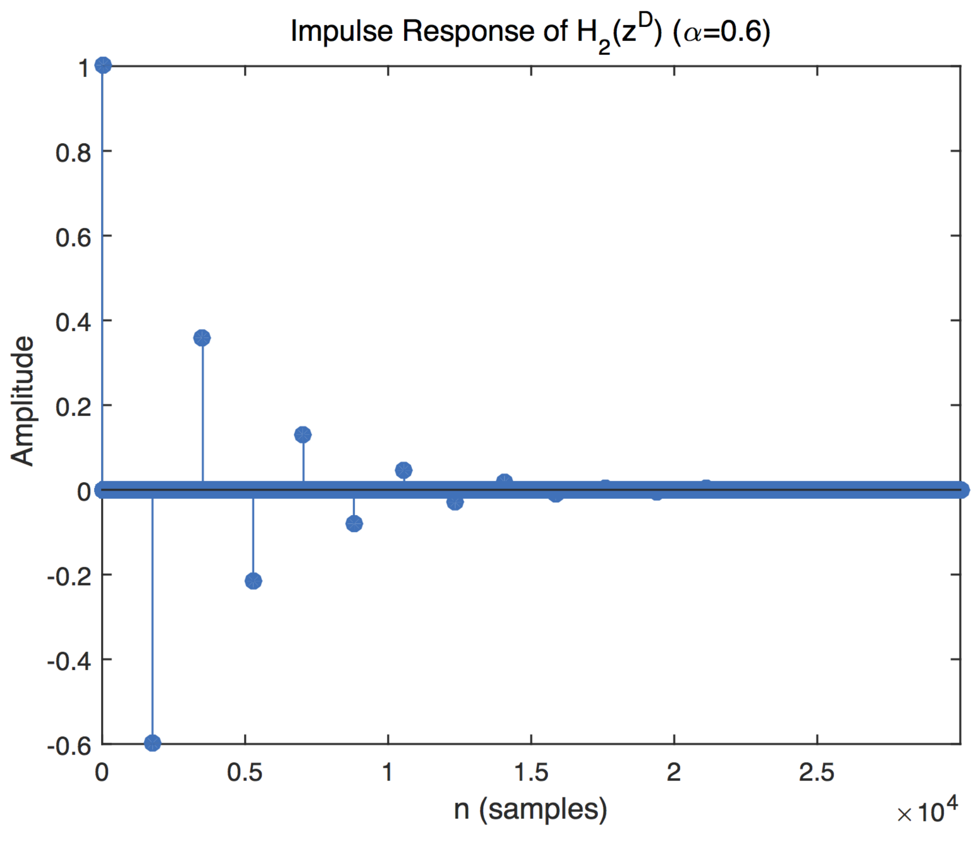

Figure 3: Impulse Response $H_2(z^D)$ ($\alpha = 0.6$)

The signal generated in b) (ii) is slightly louder, especially at the beginning of the audio. As is shown in Figure.3, the impulse response of $H_2(z^D)$ consists of infinite impulses, in a relative short time (4 seconds comparing with 220ms delay), only a part of them will present. However, the calculated power of these two signals are the same when $\alpha_2 = 0.6$ and in infinitely long time.

(iv)

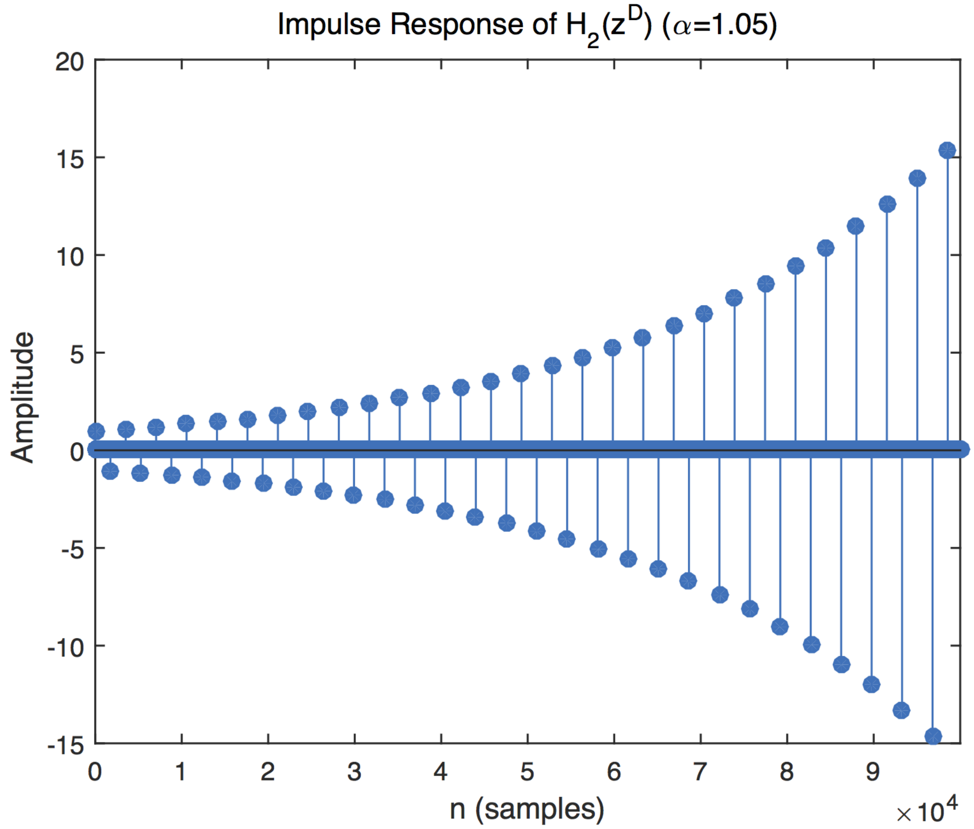

Figure 4: Impulse Response $H_2(z^D)$ ($\alpha = 1.05$)

Echoes crescendo, even louder than the input signal. As $\alpha = 1.05 > 1$, the pole is located outside the unit circle, hence the filter is **unstable**. As is shown in Figure.4, the impulse response diverges.

(v)

Time domain representation

$$
\begin{aligned}
y[n] &= x[n] + (-\alpha)x[n-D] + (-\alpha)^2 x[n-2D] + (-\alpha)^3 x[n-3D] + \cdots \\
y[n-D] &= x[n-D] + (-\alpha)x[n-2D] + (-\alpha)^2 x[n-3D] + (-\alpha)^3 x[n-4D] + \cdots \\
y[n] &= x[n] - \alpha y[n-D]
\end{aligned}
$$

The output sound has multiple echoes comparing with the first filter.

B c)

(i)

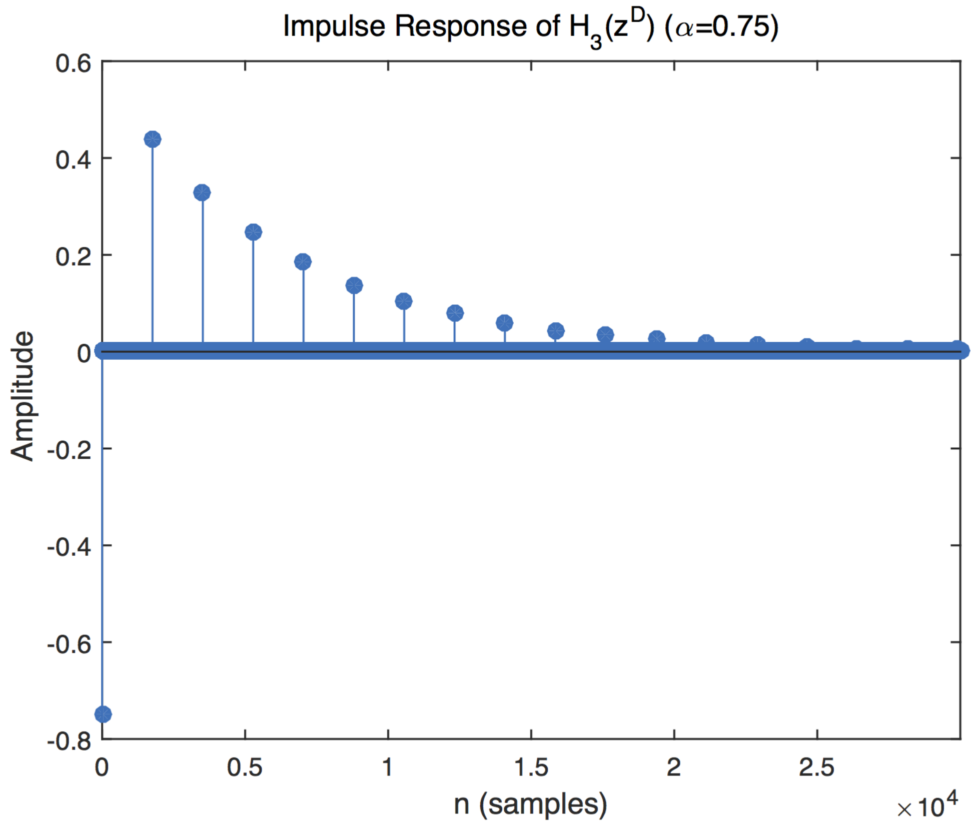

Figure 5: Impulse Response $H_3(z^D)$ ($\alpha = 0.75$)

It can be clearly seen that output signal $y[n]$ is the superposition of the original signal: $x[n]$ and an infinite sequence of delayed and attenuated signals. Hence, $H_3(z^D)$ can be used as an echo filter.
The magnitude response of $H_3(z^D)$ is frequency-invariant, i.e. $|H_3(e^{j\omega})| = 1$.

By comparing Figure.2 and Figure.5, it can be concluded that the magnitude of impulses(except the first negative impulse) decay much slower in $H_3(z^D)$. In other words, the echo density is higher than the counterparts generated by $H_1(z^D)$ and $H_2(z^D)$. Therefore, $H_3(z^D)$ gives less coloration of the sound.

(ii)

$\frac{D}{F_s} = 220 \text{ ms} \longrightarrow D = 220 \text{ ms} \times 8 \text{ kHz} = 1760 \text{ samples}$

This echo generator sounds more natural than the first two and of less coloration. But we still cannot get a smooth sounding reverberation due to obvious separated echoes.

(iii)

Time domain representation

$$
\begin{aligned}
H_3(z^D) &= \frac{z^{-D} - \alpha}{1 - \alpha z^{-D}} \\
Y(z) &:= H_3(z^D)X(z) \\
Y(z) - \alpha z^{-D}Y(z) &= z^{-D}X(z) - \alpha X(z) \\
Y(z) &= z^{-D}X(z) - \alpha X(z) + \alpha z^{-D}Y(z) \\
y[n] &= x[n-D] - \alpha x[n] + \alpha y[n-D]
\end{aligned}
$$

B d)

(i)

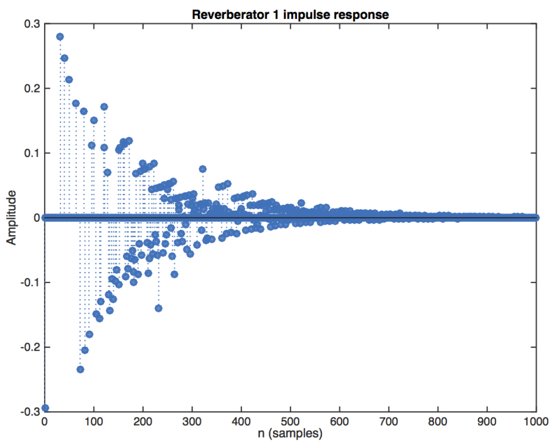

Figure 6: Reverberator 1 impulse response

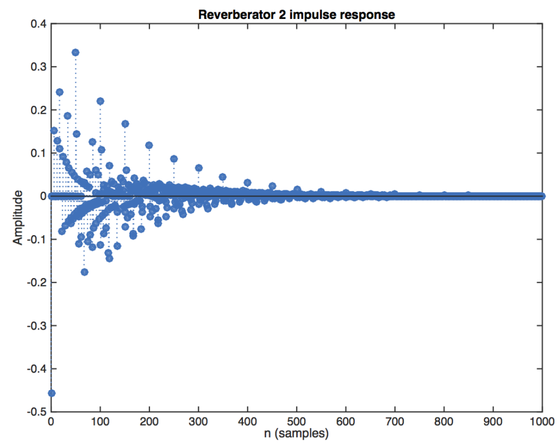

Figure 7: Reverberator 2 impulse response

Reverberator 2 is better. The impulse density is more intensive so that adjacent echoes cannot be differentiated.

(ii)

If $D_{1,2,3}$ are rounded to nearest prime numbers, their echoes will be $nD_{1,2,3}$ and the overlap probability among nD_1, nD_2 and nD_3 is much less than non-prime $D_{1,2,3}$. In other words, the impulse response is much more dense than before.

(iii)

$$
\begin{aligned}
D_1 &= 50\text{ ms} \times 8\text{ kHz} = 400 \approx 401 \\
D_2 &= 40\text{ ms} \times 8\text{ kHz} = 320 \approx 317 \\
D_3 &= 32\text{ ms} \times 8\text{ kHz} = 256 \approx 257
\end{aligned}
$$

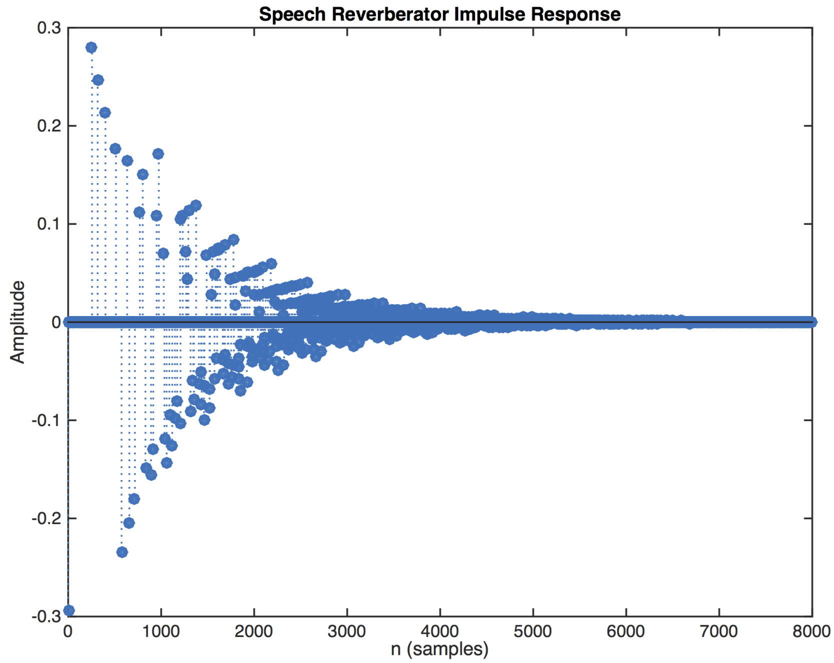

Figure 8: Impulse response

Comparing Figure.6 and Figure.8, the impulse density increases and the sound is more natural.

C c)

AddSinus()

$F = 550$ Hz

```
#define SAMPLE_RATE 24000.0
#define PI 3.14159265359

static float A = 3E8;            // noise amplitude
static float F = 550;            // Central frequency

void AddSinus(void) {
    static int index = 0;
    // declaring as static keeps 'index' between invocations

    float disturbance = A * cos(F * 2 * PI * index / SAMPLE_RATE);
    // sinusoid wave

    index++;
    index = index % (int)SAMPLE_RATE;
    // 'index' counts from 0 up to SAMPLE_RATE and starts at 0 again.
    // in case of 'index' overflow

    LeftInputCorrupted = LeftInput + disturbance;
    RightInputCorrupted = RightInput + disturbance;
}
```

FilterCoeff()

$$\omega_0 = \frac{2\pi \times 550 \text{ Hz}}{24 \times 10^3 \text{ sample/s}} = 0.143990 \text{ rad/sample} \tag{8}$$

$$H_{BS}(z) = \frac{1+\alpha}{2} \frac{1 - 2\beta z^{-1} + z^{-2}}{1 - \beta(1+\alpha)z^{-1} + \alpha z^{-2}} \tag{9}$$

where $\beta = \cos(\omega_0) = \mathbf{0.989651}$.

Poles at $re^{\pm j\phi}$, a stable system requires $r = \sqrt{\alpha} < 1$, i.e. $0 < \alpha < 1$.

$$B_w = \cos^{-1}(\frac{2\alpha}{1+\alpha^2}) \tag{10}$$

Given $0 < \alpha < 1$

$$\alpha = \frac{1}{\cos(B_w)} - \sqrt{\frac{1}{(\cos(B_w))^2} - 1} \tag{11}$$

```c
#define SAMPLE_RATE 24000.0
#define PI 3.14159265359

static float BW = 0.1 * PI;     // Bandwidth
static float F = 550;           // Central frequency

static double alpha, beta;      // filter coefficients
static double a1, a2;
static double b0, b1, b2;

void FilterCoeff(void) {
    beta = cos(F * 2 * PI / SAMPLE_RATE);

    double cosine = cos(BW);
    alpha = 1/cosine - sqrt(1/(cosine*cosine) - 1);

    printf("alpha=%f\nbeta=%f\n", alpha, beta);

    double coefficient = (1 + alpha) / 2;
    a1 = -beta * (1 + alpha);
    a2 = alpha;
    b0 = coefficient;
    b1 = -2 * beta * coefficient;
    b2 = coefficient;
}
```

When $B_w = 0.1\pi$, from Eq. 11

$$\alpha = 0.726543 \tag{12}$$

When $B_w = 0.01\pi$, from Eq. 11

$$\alpha = 0.969067 \tag{13}$$

When $B_w = 0.0025\pi$, from Eq. 11

$$\alpha = 0.992177 \tag{14}$$

Time domain representation

$$
\begin{aligned}
H_{BS}(z) &= \frac{1+\alpha}{2}\frac{1-2\beta z^{-1}+z^{-2}}{1-\beta(1+\alpha)z^{-1}+\alpha z^{-2}} \\
Y(z) &:= H_{BS}(z)X(z) \\
(1-\beta(1+\alpha)z^{-1}+\alpha z^{-2})Y(z) &= \frac{1+\alpha}{2}(1-2\beta z^{-1}+z^{-2})X(z) \\
y[n]-\beta(1+\alpha)y[n-1]+\alpha y[n-2] &= \frac{1+\alpha}{2}(x[n]-2\beta x[n-1]+x[n-2])
\end{aligned}
$$

$$
\begin{aligned}
y[n] &= \frac{1+\alpha}{2}(x[n]-2\beta x[n-1]+x[n-2])+\beta(1+\alpha)y[n-1]-\alpha y[n-2] \\
&= b_0x[n]+b_1x[n-1]+b_2x[n-2]-a_1y[n-1]-a_2y[n-2]
\end{aligned}
$$

$$
\begin{aligned}
b_0 &= \frac{1+\alpha}{2} \\
b_1 &= 2\beta\cdot\frac{1+\alpha}{2} \\
b_2 &= \frac{1+\alpha}{2} \\
a_1 &= -\beta(1+\alpha) \\
a_2 &= \alpha
\end{aligned}
$$

C d)

When smaller bandwidths ($0.1\pi \to 0.01\pi \to 0.0025\pi$) are adopted, the filtered signal sounds clearer.

Appendix

SPWS2-echo/_LabTasks.c

```c
#include "SPWS2-echo.h"

// Input samples
float LeftInput;
float RightInput;

// Output samples
float loa, lob, loc;

// Declare any global variables you need
#define N 1760

static float alpha13 = 0.75;
static float alpha2 = 0.6;

static float input[N] = {0.0};
static float outputB[N] = {0.0};
static float outputC[N] = {0.0};

static int current = 0;

void EchoFilter(void) {
    /* TODO: Implement echo filter (a) */
    loa = LeftInput + alpha13 * input[current];
    // input[current] is input[current] in last sampling period because it has not been
        updated.

    /* TODO: Implement echo filter (b) */
    outputB[current] = LeftInput - alpha2 * outputB[current];
    // the second outputB[current] is outputB[current] in last sampling period. outputB[
        current] is update by this line.
    lob = outputB[current];

    /* TODO: Implement echo filter (c) */
    outputC[current] = input[current] - alpha13 * LeftInput + alpha13 * outputC[current];
    // input[current] is input[current] in last sampling period because it has not been
        updated.
    // the second outputC[current] is outputC[current] in last sampling period. outputC[
        current] is update by this line.
    loc = outputC[current];

    /* update input[current] */
    input[current] = LeftInput;

    current++;
    current = current % N;
}
```

SPWS2-notch/_LabTasks.c

```c
#include "SPWS2-notch.h"

#define SAMPLE_RATE 24000.0
#define PI 3.14159265359

// Input samples
float LeftInput;
float RightInput;

// Corrupted samples
float LeftInputCorrupted;
float RightInputCorrupted;

// Filtered samples
float LeftOutputFiltered;
float RightOutputFiltered;


// TODO: 0. Define your own global coefficients for filtering
#define BUFFER_SIZE     2
//define the buffer size

#define INDEX(CURRENT)  ((CURRENT) + BUFFER_SIZE) % BUFFER_SIZE
// if an index is negative, a specified position from the end of the array will be returned.
// e.g. given an array x[8], x[INDEX(-1)] and x[INDEX(7)] both refer to x[7].

static float A = 1E8;           // noise amplitude
static float BW = 0.1 * PI;     // Bandwidth
static float F = 550;           // Central frequency

static double alpha, beta;      // filter coefficients
static double a1, a2;
static double b0, b1, b2;

static float xBufferL[BUFFER_SIZE] = {0.0};
static float xBufferR[BUFFER_SIZE] = {0.0};
// input buffer (Left and Right)

static float yBufferL[BUFFER_SIZE] = {0.0};
static float yBufferR[BUFFER_SIZE] = {0.0};
// output buffer (Left and Right)


void FilterCoeff(void) {
    // TODO: 1. Initialise the filter coefficients
    //   You should write this function so the filter centre frequency and
    //   bandwidth can be easily changed.

    beta = cos(F * 2 * PI / SAMPLE_RATE);

    double cosine = cos(BW);
    alpha = 1/cosine - sqrt(1/(cosine*cosine) - 1);

    printf("alpha=%f\nbeta=%f\n", alpha, beta);

    double coefficient = (1 + alpha) / 2;
    a1 = -beta * (1 + alpha);
    a2 = alpha;
    b0 = coefficient;
    b1 = -2 * beta * coefficient;
```

```
    b2 = coefficient;
}


void AddSinus(void) {
    // TODO: 2. Add the sinusoidal disturbance to the input samples

    static int index = 0;
    // declaring as static keeps 'index' between invocations

    float disturbance = A * cos(F * 2 * PI * index / SAMPLE_RATE);
    // sinusoid wave

    index++;
    index = index % (int)SAMPLE_RATE;
    // 'index' counts from 0 up to SAMPLE_RATE and starts at 0 again.
    // in case of 'index' overflow

    LeftInputCorrupted = LeftInput + disturbance;
    RightInputCorrupted = RightInput + disturbance;
}

void NotchFilter(void) {
    // TODO: 3. Filter the corrupted samples

    static int current = 0;

    LeftOutputFiltered = b0 * LeftInputCorrupted + b1 * xBufferL[INDEX(current-1)] + b2 *
        xBufferL[INDEX(current-2)] - a1 * yBufferL[INDEX(current-1)] - a2 * yBufferL[INDEX(
        current-2)];
    RightOutputFiltered = b0 * RightInputCorrupted + b1 * xBufferR[INDEX(current-1)] + b2 *
        xBufferR[INDEX(current-2)] - a1 * yBufferR[INDEX(current-1)] - a2 * yBufferR[INDEX(
        current-2)];

    xBufferL[current] = LeftInputCorrupted;
    xBufferR[current] = RightInputCorrupted;

    yBufferL[current] = LeftOutputFiltered;
    yBufferR[current] = RightOutputFiltered;

    current++;
    current = current % BUFFER_SIZE;
}
```

B a) (ii)

```
clear;

[x, fs] = audioread('speech.wav');
timestamp = clock;
sound(x, fs);
disp('Playing original signal');

D = 1760;
alpha = 0.75;

num = zeros(1, D+1);
num(1) = 1;
num(D+1) = alpha;

den = 1;

figure;
impz(num, den, 2000);
title('Impulse Response of H_1(z^D) (\alpha=0.75)');

outputA2 = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
end

sound(outputA2, fs);
disp('Playing signal generated in a) (ii)');
```

B b) (ii)

```
clear;

[x, fs] = audioread('speech.wav');
timestamp = clock;
sound(x, fs);
disp('Playing original signal');

D = 1760;
alpha = 0.75;

% b) (ii)
den = zeros(1, D+1);
den(1) = 1;
den(D+1) = alpha;

num = 1;

figure;
impz(num, den, 30000);
title('Impulse Response of H_2(z^D) (\alpha=0.75)');

outputB2 = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
```

```
end

timestamp = clock;
sound(outputB2, fs);
disp('Playing signal generated in b) (ii)');

% a) (ii)
num = zeros(1, D+1);
num(1) = 1;
num(D+1) = alpha;

den = 1;

outputA2 = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
end

sound(outputA2, fs);
disp('Playing signal generated in a) (ii)');
```

B b) (iii)

```
clear;

[x, fs] = audioread('speech.wav');
timestamp = clock;
sound(x, fs);
disp('Playing original signal');

% b) (ii)
D = 1760;
alpha = 0.6;

den = zeros(1, D+1);
den(1) = 1;
den(D+1) = alpha;

num = 1;

figure;
impz(num, den, 30000);
title('Impulse Response of H_2(z^D) (\alpha=0.6)');

outputB2 = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
end

timestamp = clock;
sound(outputB2, fs);
disp('Playing signal generated in b) (iii)');

% a) (ii)
D = 1760;
alpha = 0.75;
```

```
num = zeros(1, D+1);
num(1) = 1;
num(D+1) = alpha;

den = 1;

outputA2 = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
end

sound(outputA2, fs);
disp('Playing signal generated in a) (ii)');
```

B b) (iv)

```
clear;

[x, fs] = audioread('speech.wav');
timestamp = clock;
sound(x, fs);

D = 1760;
alpha = 1.05;

den = zeros(1, D+1);
den(1) = 1;
den(D+1) = alpha;

num = 1;

figure;
impz(num, den, 1E5);
title('Impulse Response of H_2(z^D) (\alpha=1.05)');

output = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
end

sound(output, fs);
```

B c) (ii)

```
clear;

[x, fs] = audioread('speech.wav');
t1 = clock;
sound(x, fs);

D = 1760;
alpha = 0.75;

num = zeros(1, D+1);
num(1) = -alpha;
```

```
num(D+1) = 1;

den = zeros(1, D+1);
den(1) = 1;
den(D+1) = -alpha;

figure;
impz(num, den, 30000);
title('Impulse Response of H_3(z^D) (\alpha=0.75)');

output = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, t1);
if (time > 0)
    pause(time);
end

sound(output, fs);
```

B d) MATLAB function

```
function [num, den] = Bd_function(D, alpha)
    num1 = zeros(1, D(1)+1);
    num1(1) = -alpha(1);
    num1(D(1)+1) = 1;
    den1 = zeros(1, D(1)+1);
    den1(1) = 1;
    den1(D(1)+1) = -alpha(1);

    num2 = zeros(1, D(2)+1);
    num2(1) = -alpha(2);
    num2(D(2)+1) = 1;
    den2 = zeros(1, D(2)+1);
    den2(1) = 1;
    den2(D(2)+1) = -alpha(2);

    num3 = zeros(1, D(3)+1);
    num3(1) = -alpha(3);
    num3(D(3)+1) = 1;
    den3 = zeros(1, D(3)+1);
    den3(1) = 1;
    den3(D(3)+1) = -alpha(3);

    num = conv(num1, num2);
    num = conv(num, num3);
    den = conv(den1, den2);
    den = conv(den, den3);
end
```

B d) (i)

```
clear;
close all;

% Reverberator 1
D1 = 50;
D2 = 40;
D3 = 32;

alpha1 = 0.7;
alpha2 = 0.665;
```

```
alpha3 = 0.63175;

[num, den] = Bd_function([D1 D2 D3], [alpha1 alpha2 alpha3]);

[h,t] = impz(num, den, 1000);
figure;
stem(t, h, 'filled', 'LineStyle', ':');
xlabel('n (samples)');
ylabel('Amplitude');
title('Reverberator 1 impulse response');

% Reverberator 2
D1 = 50;
D2 = 17;
D3 = 6;

alpha1 = 0.7;
alpha2 = 0.77;
alpha3 = 0.847;

[num, den] = Bd_function([D1 D2 D3], [alpha1 alpha2 alpha3]);

[h,t] = impz(num, den, 1000);
figure;
stem(t, h, 'filled', 'LineStyle', ':');
xlabel('n (samples)');
ylabel('Amplitude');
title('Reverberator 2 impulse response');
```

B d) (iii)

```
clear;

[x, fs] = audioread('speech.wav');
timestamp = clock;
sound(x, fs);

D1 = 50E-3;     % 50ms
D2 = 40E-3;     % 40ms
D3 = 32E-3;     % 32ms

D = [D1 D2 D3];
D = D * fs;

for i=1:3
    fprintf('D%d = %d ', i, D(i));
    D(i) = max(primes( D(i)*2 - max(primes(D(i))) ));
    % round the delays in samples to the nearest prime number
    fprintf('~= %d\n', D(i));
end

alpha1 = 0.7;
alpha2 = 0.665;
alpha3 = 0.63175;

[num, den] = Bd_function(D, [alpha1 alpha2 alpha3]);

[h,t] = impz(num, den, 8E3);
figure;
stem(t, h, 'filled', 'LineStyle', ':');
xlabel('n (samples)');
ylabel('Amplitude');
```

```
title('Speech Reverberator Impulse Response');

output = filter(num, den, x);

% wait for the sound to finish
time = length(x)/fs - etime(clock, timestamp);
if (time > 0)
    pause(time);
end

sound(output, fs);
```