1. (30 pts) Convolution sum and echo time estimation noise remover for echo time estimation

(a) (5 pts) Please elaborate why the impulse response of the noise remover is design in the form presented in the given sample codes (see **h** in the sample codes of Part 1).

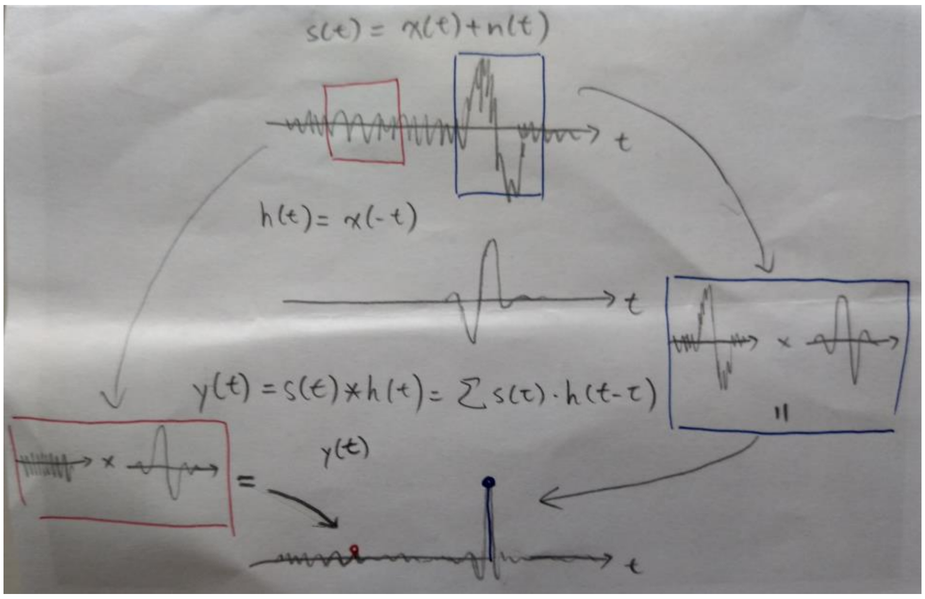
From lecture note (slide 59, Topic2_LTISystems_Part1_HandWriting0331_2021 .pdf), we know that doing convolution to ***x*** and ***h*** is equivalent to projecting ***x*** onto time reversal of ***h***, which can calculate the similarity between them to check whether they have same feature.

We have already known the signal that sonar transmitted is a chirp signal, which is the feature of the desired signal. Thus, we set the time reversal of chirp signal as ***h***.

```
h = fliplr(x);
```

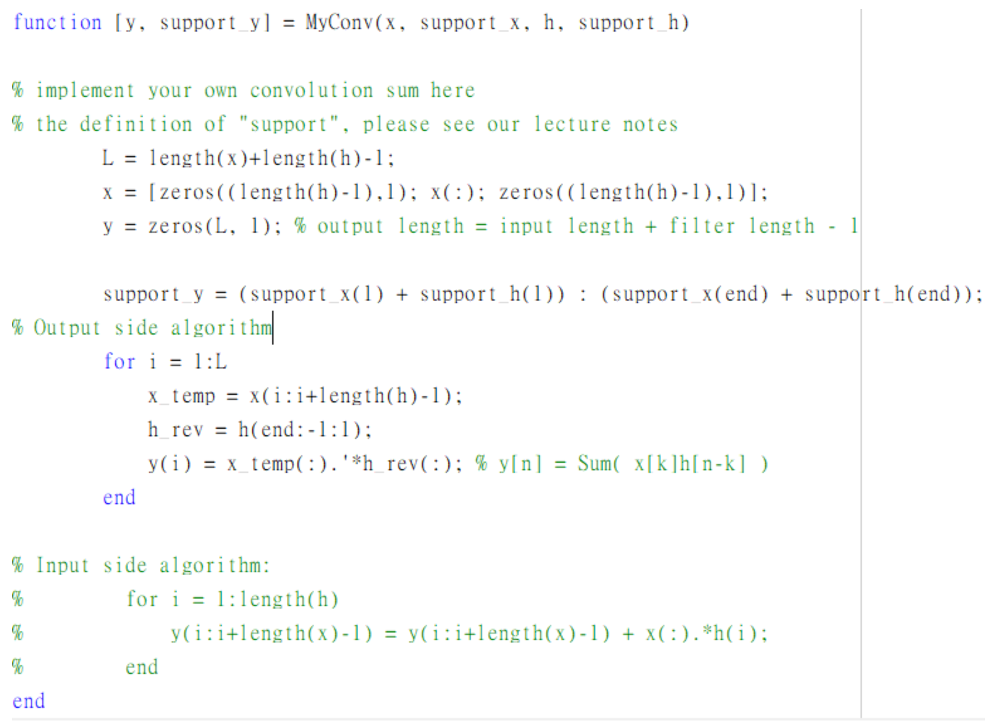
By doing so, the time reversal of ***h*** will be exactly the same as the signal feature. Therefore, you will get a small value when projecting noise part onto time reversal of ***h***, and when projecting the echo part, which has the feature of chirp signal as same as the time reversal of ***h***, you will get a larger value obviously. So, you can find the echo time of the signal.

We can understand the concept in this way.



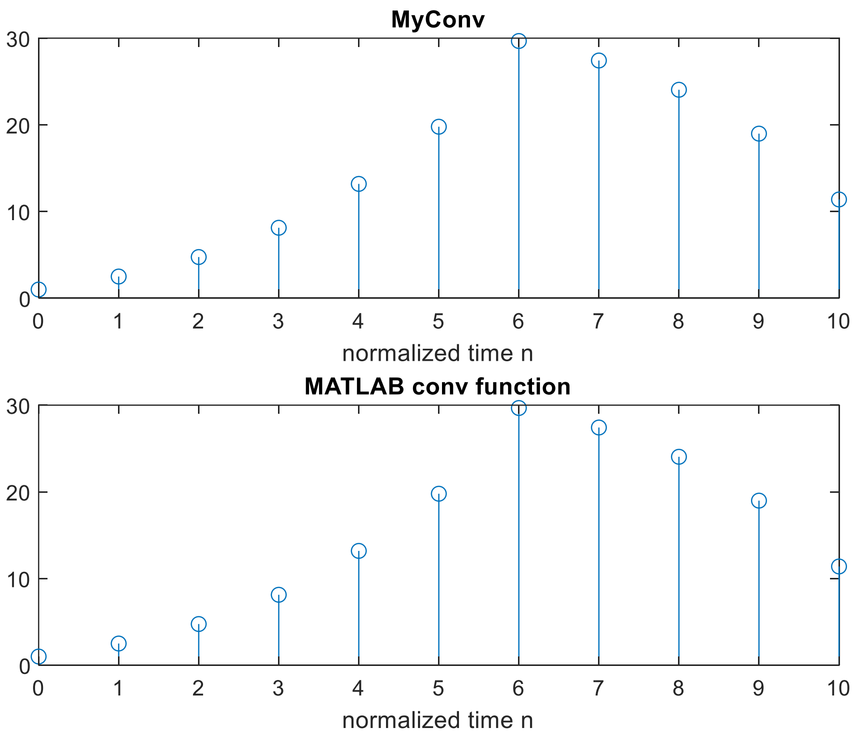
Such method is also called matched filtering. (slide 51 52, Topic2_LTISystems_ Part2_HandWriting0407_2021.pdf)

(b) (10 pts) Implement your own convolution sum function — **MyConv()** based on the given function protoype defined in the provided **MyConv.m**. Note that please use Example 2.4 (slide 50, Topic2_ LTISystems_Part1_Han dWriting0331 _2021 .pdf) to valid your own MyConv() and also compare with the results from MATLAB built in function **conv()**.

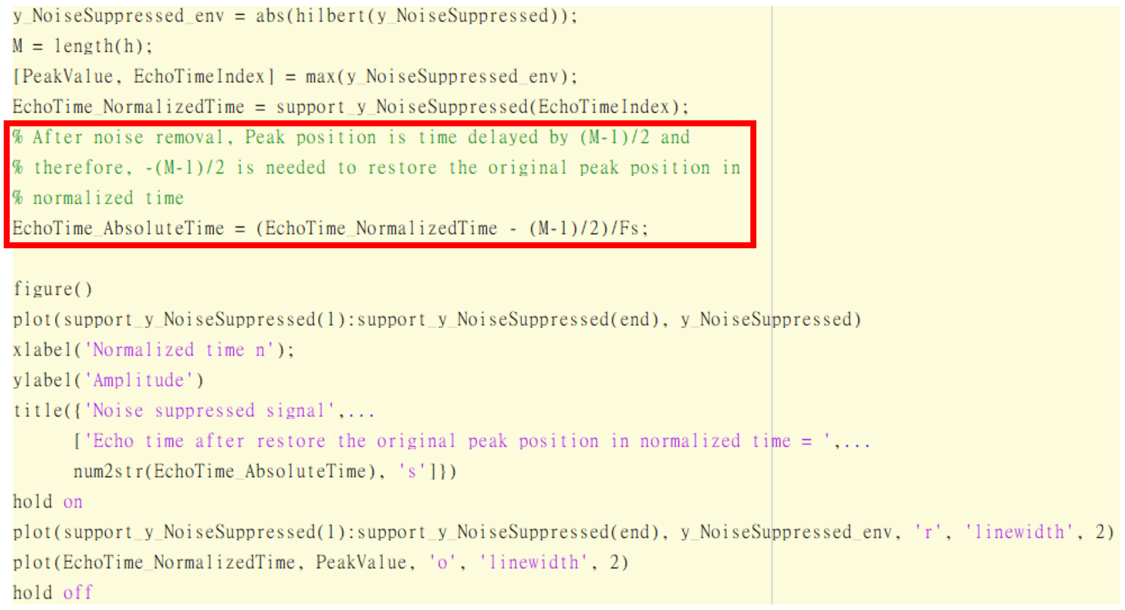


```
function [y, support_y] = MyConv(x, support_x, h, support_h)

% implement your own convolution sum here
% the definition of "support", please see our lecture notes
        L = length(x)+length(h)-1;
        x = [zeros((length(h)-1),1); x(:); zeros((length(h)-1),1)];
        y = zeros(L, 1); % output length = input length + filter length - 1

        support_y = (support_x(1) + support_h(1)) : (support_x(end) + support_h(end));
% Output side algorithm
        for i = 1:L
            x_temp = x(i:i+length(h)-1);
            h_rev = h(end:-1:1);
            y(i) = x_temp(:).'*h_rev(:); % y[n] = Sum( x[k]h[n-k] )
        end

% Input side algorithm:
%         for i = 1:length(h)
%             y(i:i+length(x)-1) = y(i:i+length(x)-1) + x(:).*h(i);
%         end
end
```

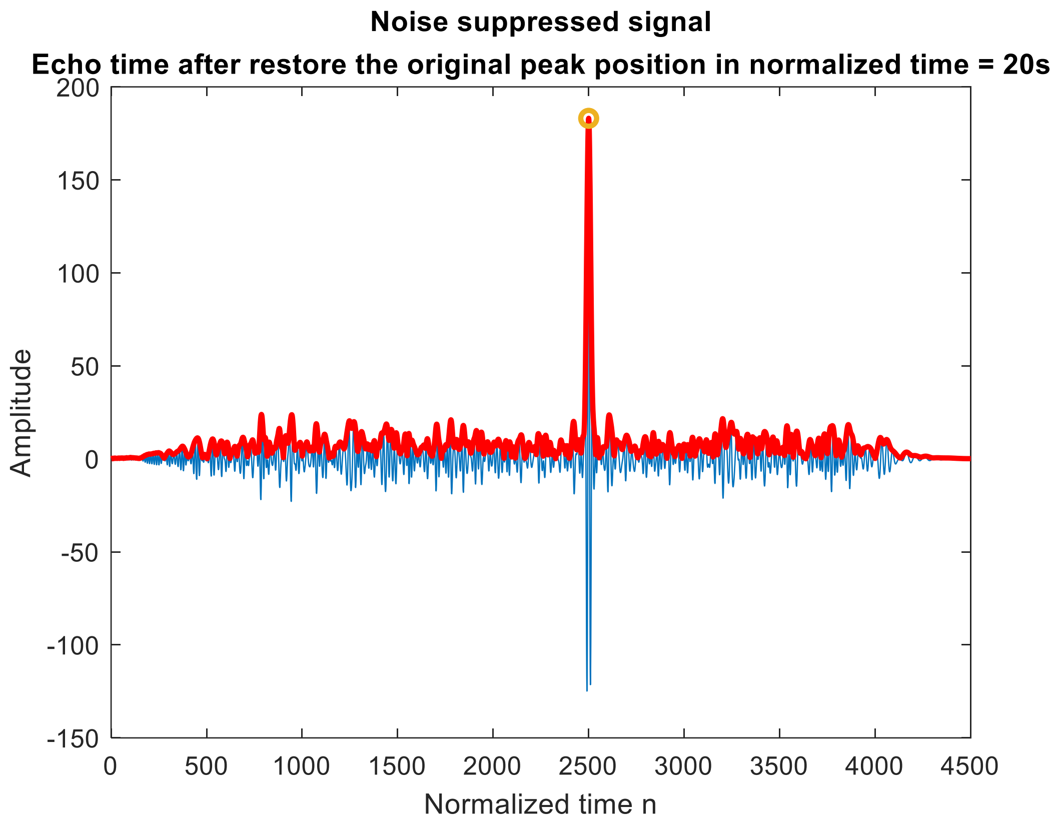
Validation by textbook example 2.4:



(c) (10 pts) Run the noise remover with **y** as its input, plot the noise suppressed signal as a function of absolute time, and then estimate the echo time from the noise suppressed signal.



```
y_NoiseSuppressed_env = abs(hilbert(y_NoiseSuppressed));
M = length(h);
[PeakValue, EchoTimeIndex] = max(y_NoiseSuppressed_env);
EchoTime_NormalizedTime = support_y_NoiseSuppressed(EchoTimeIndex);
% After noise removal, Peak position is time delayed by (M-1)/2 and
% therefore, -(M-1)/2 is needed to restore the original peak position in
% normalized time
EchoTime_AbsoluteTime = (EchoTime_NormalizedTime - (M-1)/2)/Fs;

figure()
plot(support_y_NoiseSuppressed(1):support_y_NoiseSuppressed(end), y_NoiseSuppressed)
xlabel('Normalized time n');
ylabel('Amplitude')
title({'Noise suppressed signal',...
     ['Echo time after restore the original peak position in normalized time = ',...
     num2str(EchoTime_AbsoluteTime), 's']})
hold on
plot(support_y_NoiseSuppressed(1):support_y_NoiseSuppressed(end), y_NoiseSuppressed_env, 'r', 'linewidth', 2)
plot(EchoTime_NormalizedTime, PeakValue, 'o', 'linewidth', 2)
hold off
```

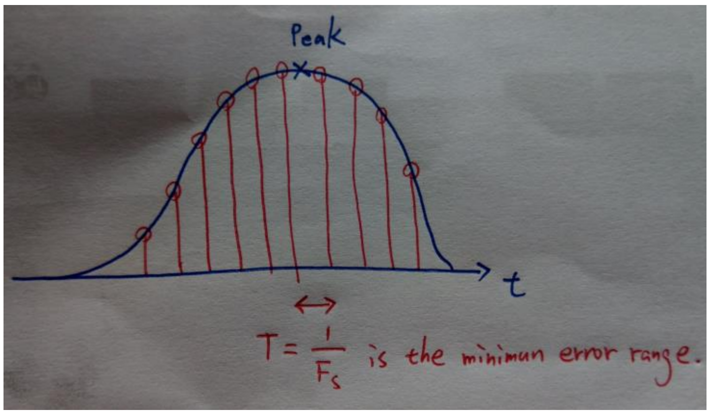


Notice that the normalized time of peak position should minus (M-1)/2 to get the correct echo time because the output of the designed LTI system will be time delayed by (M-1)/2 points. (red block in the above code)

(slide 73, Topic2_LTISystems_Part1_HandWriting0331_2021.pdf)

(d) (5 pts) Please tell what is the minimum achievable error (in seconds) of the estimated echo time and justify your answer.

If other error sources, e.g., errors caused by noises, can be perfectly ruled out, the minimum achievable error is according to the **sampling interval = 1/Fs = 0.01sec.** As the sampling interval decrease, the minimum achievable error become smaller.



2. (70 pts) Reverberation implementation

(a) (10 pts) Derive the impulse response of this reverberator, and then use the MATLAB function **filter()** to compute and plot the impulse responses of this reverberator to verify your derivation , where a = 0.7, and D = 5.

Ans:

① Proof (8 pts)

$$\begin{aligned}
&\mathrm{y[n]} = \mathrm{ay[n-D]} + \mathrm{x[n]} \\
&= a\{ay[(n-D)-D] + x[(n-D)]\} + x[n] \\
&= a^2y[n-2D] + ax[n-D] + x[n] \\
&= a^2\{ay[(\mathrm{n}-2\mathrm{D})-D] + x[(n-2D)]\} + ax[n-D] + x[n] \\
&= a^3y[n-3D] + a^2x[n-2D] + ax[n-D] + x[n] \\
&= \cdots doing\ iteration\ many\ times
\end{aligned}$$

$$\Rightarrow \mathrm{y[n]} = \sum_{k=0}^{\infty} a^k x[n-kD]$$

When $\mathrm{x[n]} = \delta[\mathrm{n}]$ *is impulse, then the impulse response will be*

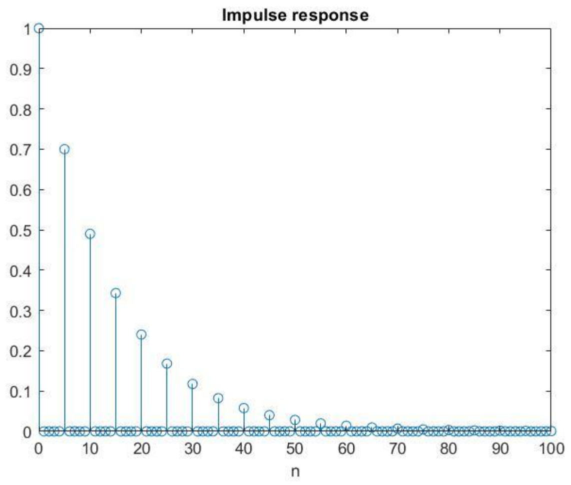
$$h[n] = \sum_{k=0}^{\infty} a^k \delta[n-kD]$$

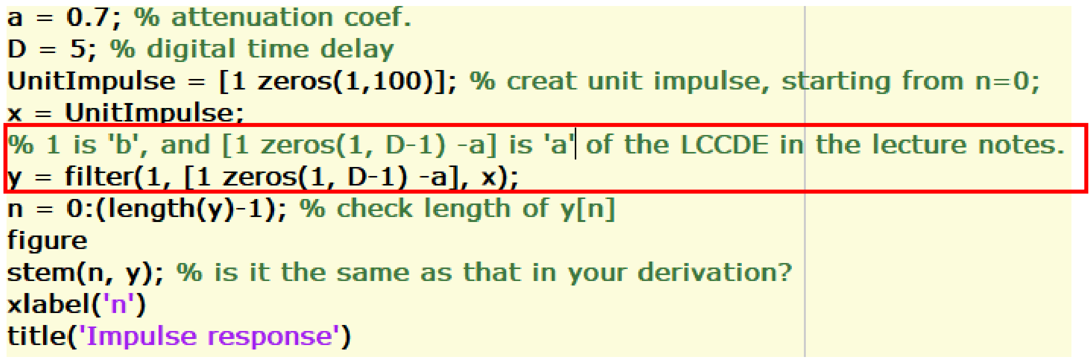
Where $\mathrm{a} = 0.7$, and $\mathrm{D} = 5$

$$\mathrm{h[n]} = \sum_{k=0}^{\infty} 0.7^k x[n-5k]$$

$$\mathrm{h[n]} = \delta[\mathrm{n}] + 0.7\delta[\mathrm{n}-5] + 0.49\delta[\mathrm{n}-10] + 0.343\delta[\mathrm{n}-15] + \cdots$$

② Figure (2 pts)





```
a = 0.7; % attenuation coef.
D = 5; % digital time delay
UnitImpulse = [1 zeros(1,100)]; % creat unit impulse, starting from n=0;
x = UnitImpulse;
% 1 is 'b', and [1 zeros(1, D-1) -a] is 'a' of the LCCDE in the lecture notes.
y = filter(1, [1 zeros(1, D-1) -a], x);
n = 0:(length(y)-1); % check length of y[n]
figure
stem(n, y); % is it the same as that in your derivation?
xlabel('n')
title('Impulse response')
```

(b) (10 pts) Please comment whether this reverberator can be potentially implemented in real time and under what condition of a and D this reverberator is stable. Prove them.

Ans:

① Proof (4 pts)

To implement the comb reverberator potentially in real time, the system should be a causal system.

$$y[n] = \sum_{k=0}^{\infty} a^k x[n-kD]$$

The index of input should be smaller than the output index. $n - kD \leq n$ and $h[n] = 0, \text{for } n < 0$. Because $D \geq 0$, the system output only depends on current input and past input.

Thus, the system is casual system, and it can be potentially implemented in real time.

② Proof (4 pts)

By the definition in the textbook, a LTI system is stable, in BIBO sense, if and only if h[n] is absolutely summable, which means $\sum_{m=-\infty}^{\infty} |h[n]| < \infty$.

$$h[n] = \delta[n] + 0.7\delta[n-5] + 0.49\delta[n-10] + 0.343\delta[n-15] + \cdots$$

For all-pass reverberator,

$$\sum_{n=-\infty}^{\infty} |h[n]| = |1| + |a| + |a^2| + |a^3| + \cdots = \lim_{n\to\infty} \frac{1-|a^n|}{1-|a|}$$

Because $|1| + |a| + |a^2| + |a^3| + \cdots = \lim_{n\to\infty} \frac{1-|a^n|}{1-|a|}$ is finite i.e., absolutely summable, if $|a| < 1$. The system is stable only when $|a| < 1$.

③ Discussion (2 pts)

$|a| < 1$ (1 pt)

The parameter D is nothing to do with the stability of the system, because it only controls how fast h[n] changes. (1 pt)

(c) (10 pts) Approximate this reverberator by an FIR system (see slide 46, in Topic2_LTISystems_Part2_withNotes.pdf) and implement it using your own convolution sum function **MyConv()** created in Part 1 with a = 0.7 and D = τFs where D is an integer representing the digital time delay given the continuous-time delay τ = 0.1 sec. and sampling rate Fs (in Hz). Note that you may verify your results by the MATLAB built in convolution function **conv()**. Given the sound file **Halleluyah.wav**, you can load the file, play the sound and save the output sound by using the following MATLAB codes:

```
[x, Fs] = audioread(' Halleluyah.wav');
sound(x, Fs);
audiowrite(y,Fs, ' Halleluyah_FIRecho.wav');
```

where x is the input sound (a column vector), y is the output sound (a column vector, too), and Fs is the sampling rate in samples/sec (i.e., in Hz). Please elaborate how you approximate the reverberator by an FIR system. That is, elaborate how you determine the order of the FIR system (i.e., the M in the general LCCDE in slides 21 and 22, in Topic2_LTISystems_Part2_withNotes.pdf). Also plot x and y together as a function of time and check the changes in x done by your FIR system. You may need to zoom in the plot to see the changes in details. The simplest way to see the change is to show y – x (which I call trashogram). Please tell what **initial condition (i.e., y[n] and x[n], when n < 0)** is used when you implement it using your own function **MyConv()**.

Ans:

① Decide order in FIR filter (6 pts)

$$y[n] = \sum_{k=0}^{\infty} a^k x[n-kD]$$

How to select the filter order?

In this case, a = 0.7 and assume $a^N < 0.01$ is small enough. In this case, we choose N = 13.

$$\Rightarrow 0.7^{13} \cong 0.0097 < 0.01, \text{order} = \text{ND} = 13 * \text{round}(0.1 * \text{Fs})$$
$$= 13 * \text{round}(0.1 * 8192) = 13 * 819 = 10647 = \text{M}$$

Thus

$$\text{y[n]} = \sum_{k=0}^{M} a^k x[n - kD]$$

② Initial condition

Here, the initial conditions are set to be zero (i.e., y[n]=x[n]=0, when n < 0) in MyConv().

③ Figure (4 pts)

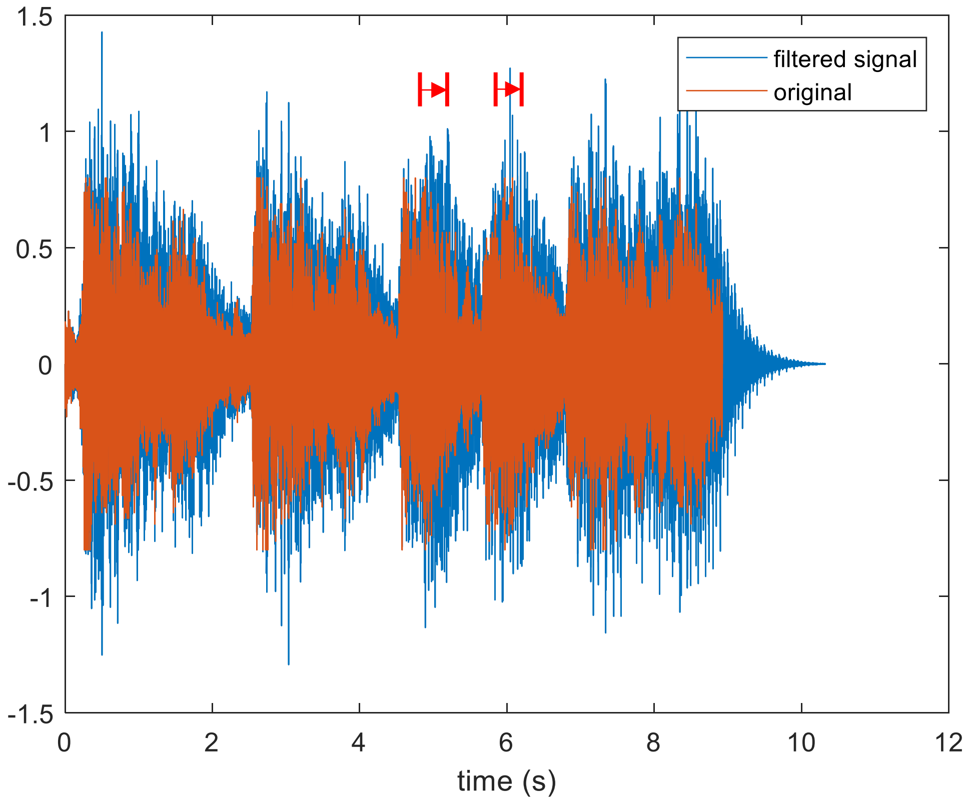


Figure: c-1

From above figures, we can find original signal and approximate FIR signal are similar. If we merge them into one figure, we can obviously find the signal of w/ FIR extended to the right, and it is echo induced by FIR filter.

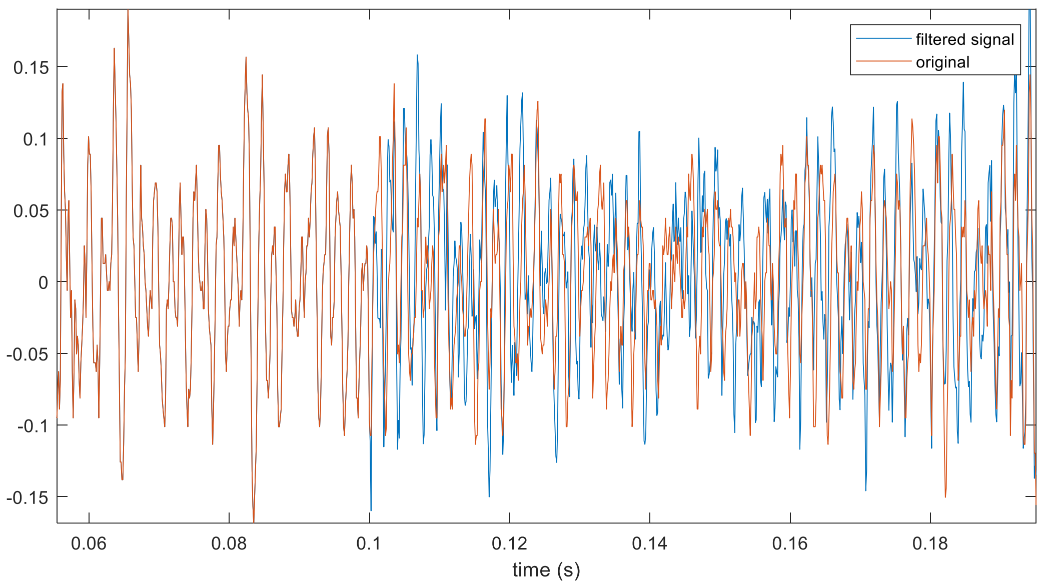


Figure: c-2

From the zooming in figure, y[n] = x[n] when normalized time n belongs to 0 to 818, and when n $\geq$ D, input and output begin to be different. Because one digital time delay D is round(0.1*fs) = 819. After that, the echo will happen.

(Note that you will find x[n] and y[n] has same value at n = 819, which is cause by y[0] = 0. y[819] = a*y[0] + x[819] = x[819].)

Trashogram of y – x

Note that you should subtract output and input with the same normalized time.

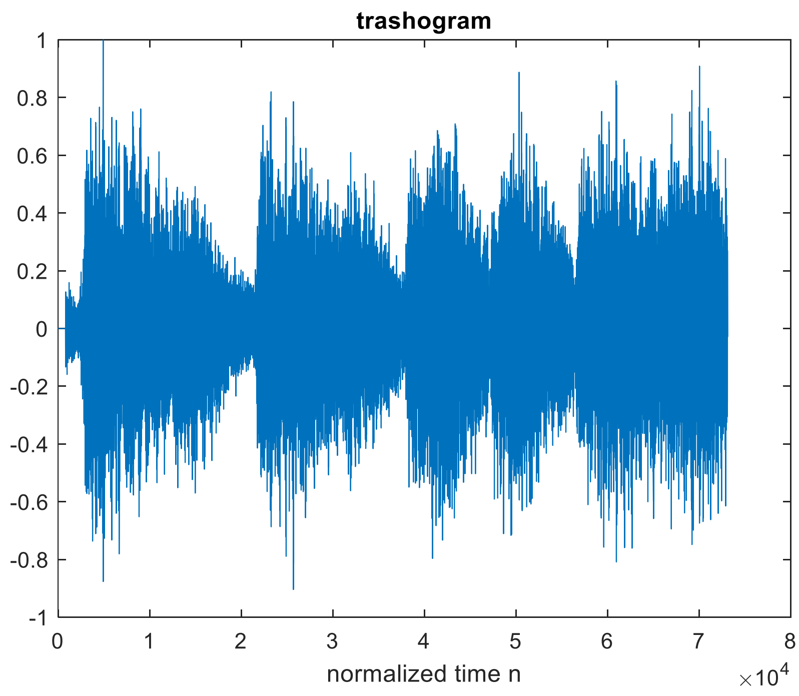


Figure: c-3

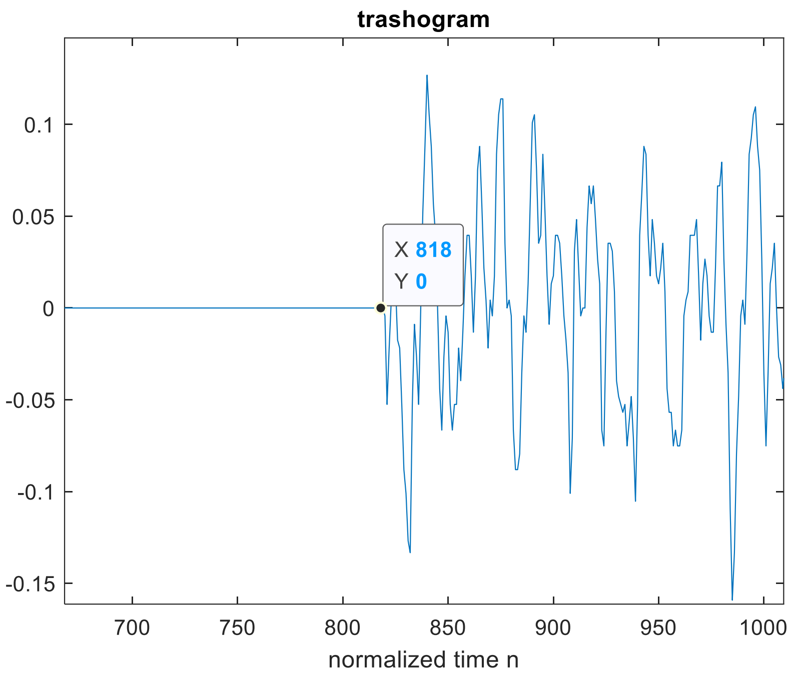


Figure: c-4

This is the zooming part of figure c-3. According to the discussion of figure c-2, input and output begin to be different when n≥D.

(d) (10 pts) Write **your own MATLAB codes (see the pseudocodes in slide 48, in Topic2_LTISystems_Part2_withNotes.pdf)** to implement this reverberator using the provided recursive LCCDE with a and D being the same values used in (c). To generate digital reverberation, we will use the same sound file in (c), and save your output sound as **Halleluyah_IIRecho.wav**. Please verify your result using MATLAB function **filter()**. Note that you have to provide the **initial condition (i.e., y[n] and x[n], when n < 0)** used in your implementation in the report. Also plot x and y together as a function of time, check the changes in x done by your IIR filter, and comment the output differences between the two different implementations in (c) and (d). The simplest way to see the difference is to show the result of (c) minus that of (d) (which I call trashogram). Note that you may need to zoom in the plot to see the changes in details. (2 plots in total, 2 pts for each plot, 2 pts for verification by function filter (this could be discussion or plot), 4 pts for other discussion)

① Here, the initial conditions are set to be zero (i.e., y[n]=x[n]=0, when n < 0).

② From figure D-1, if we merge input and output into one figure, we can obviously find the output signal extended to the right, and it is echo induced by the IIR system.

③ From figure D-2, by examined the differences between the output filtered by filter function and the output from our own method, we could make sure that our implementation is correct.

④ From figure D-3, compared to part (c), two signals are almost the same in the beginning part. However, there are small differences when the normalized time is greater than the order of FIR filter in part(c). This is because the impulse response of the FIR filter is the same as the first (M+1) terms of the impulse response of the IIR system.

⑤ Figure

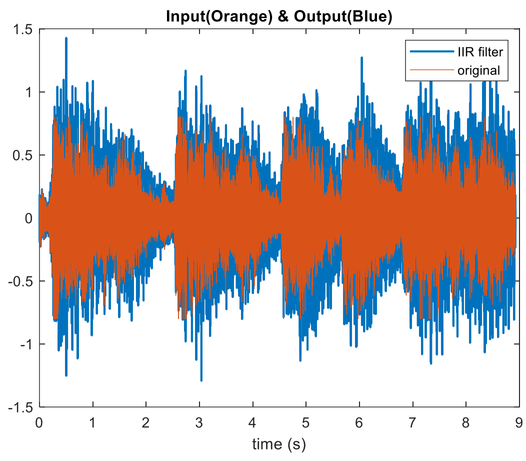


Figure: D-1

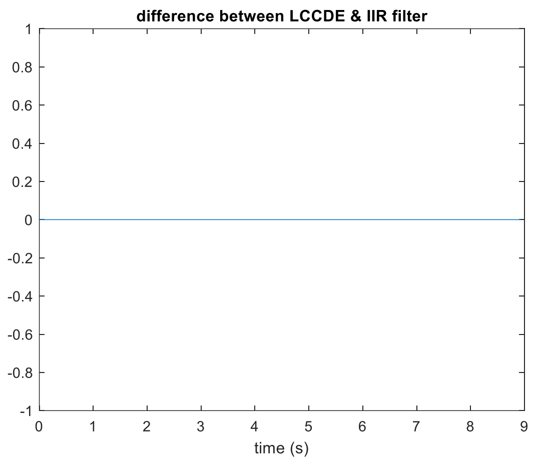


Figure: D-2

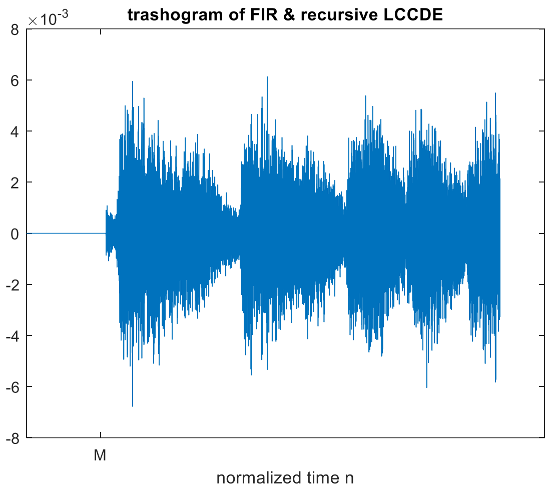


Figure: D-3

(e) (10 pts) Change the a in (d) to the values resulting in an unstable reverberator, and grasp the feeling about what the output of an unstable reverberator sounds like, also with the sound file used in d as the system input. Here you can use MATLAB function **filter()** directly if your own IIR system implantation runs too slow.

① From figure E-1 and E-2, we can observe that the tail part of the output signal amplitude of an unstable reverberator becomes larger when ***a*** increased.

② By listening to the output signal, it keeps echoing, and we are unable to distinguish the original signal from the output eventually.

③ Figure

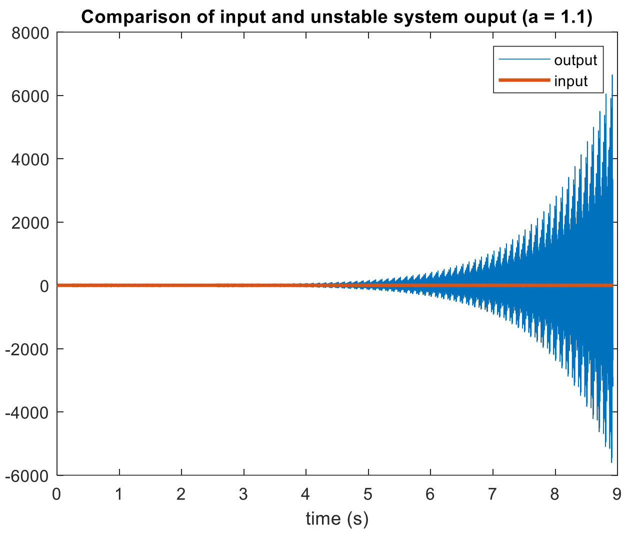


Figure E-1

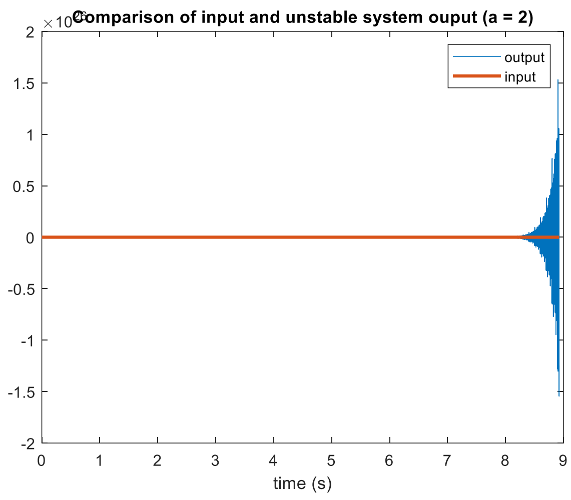


Figure E-2

(f) (10 pts) Change the initial condition (e.g., y[n] = x[n] = K, when n < 0, or y[n] and x[n] are random numbers (try MATLAB function **rand()**), n < 0. Note that you have to try to scale the initial conditions y[n] or x[n] to the values close to your input, e.g., K = the mean value of the input) which you used in (d), see and comment how the initial condition affects the results. Plot the outputs with different initial condition together as a function of time and tell the difference, also with the sound file used in (c) as the system input and the a and D the same values used in (c). (5 pts for plot, 5pts for discussion)

① From figure f-1 and figure f-2, we can observe that the initial condition will only influence the output signal at early time. Also, the larger the initial condition is, the higher the transient response is.

② Figure

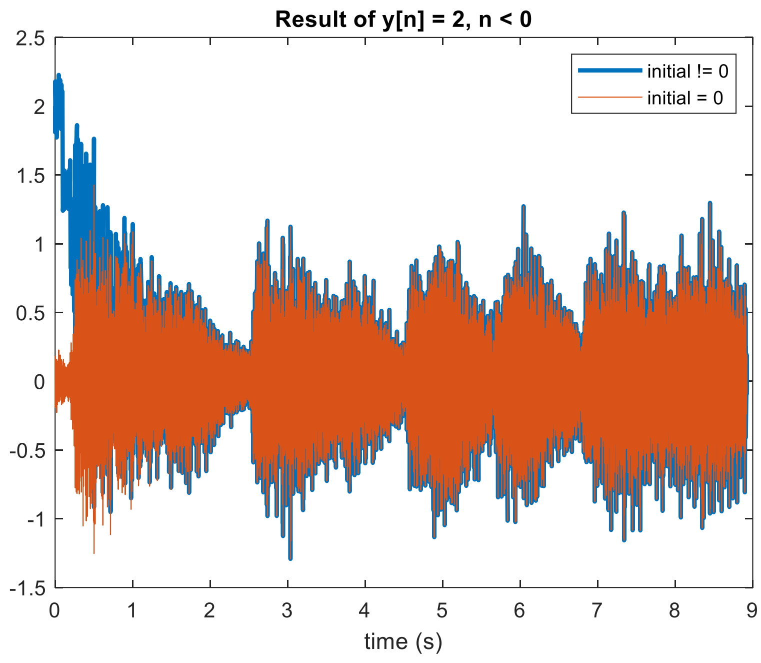


Figure: f-2

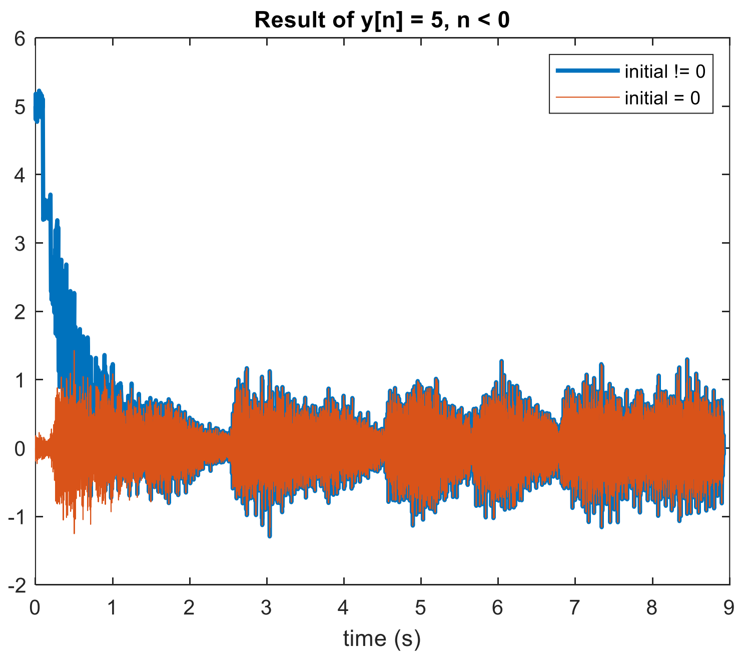


Figure: f-2

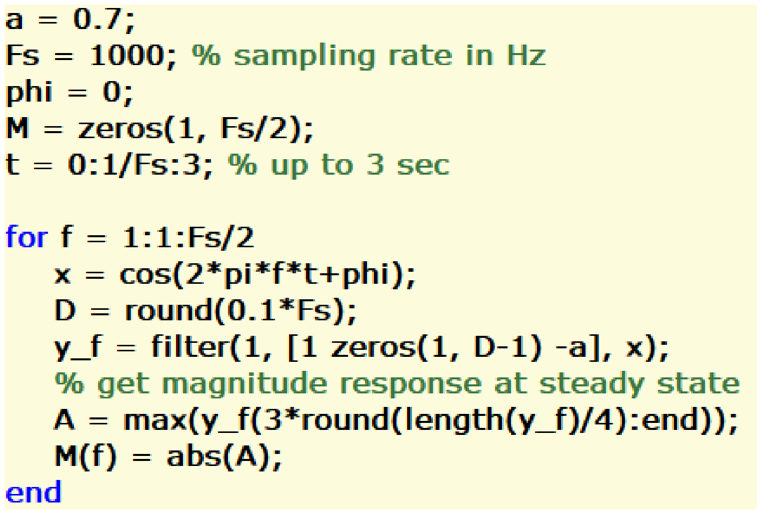
(g) (10 pts) As mentioned in the very beginning of Part 2, the implemented reverberator is a "comb reverberator". That means the system will colorize the

input sound. That is, some frequencies of the sound will be suppressed, and some frequencies will be amplified (or emphasized). Use the single tone sinusoidal signal in your computer homework #1 to figure out what frequencies (or pitches) of the sinusoidal signals will be suppressed and what frequencies (or pitches) of the sinusoidal signals will be emphasized by changing the frequency from 0 to Fs/2. For this purpose, please plot magnitude response of the system for frequency ranging from 0 to Fs/2. The definition of the magnitude response will be explained in class (see the topic of "Explanation of ComputerHW2"). Note that the a and τ (used to find D which varies with different Fs) are the same values used in (c).
(5 pts for magnitude response, 5 pts for discussion)

① In this case, we assume Fs = 1kHz. Thus, we will plot the magnitude response by changing the frequency of input cosine signal from 0~500Hz. Notice that having a larger Fs will give you a smoother magnitude response.

② According to the result of 2(a) in Computer HW1, to retain the frequency of the signal, we know that sampling frequency should be 2 times greater than the frequency of the sinusoidal signal. Therefore, the MAX frequency of sinusoidal signal is Fs/2.

③ In this case, we assume the amplitude of input cosine signal is 1, so the absolute value of amplitude of the output is the magnitude response directly.



```
a = 0.7;
Fs = 1000; % sampling rate in Hz
phi = 0;
M = zeros(1, Fs/2);
t = 0:1/Fs:3; % up to 3 sec

for f = 1:1:Fs/2
    x = cos(2*pi*f*t+phi);
    D = round(0.1*Fs);
    y_f = filter(1, [1 zeros(1, D-1) -a], x);
    % get magnitude response at steady state
    A = max(y_f(3*round(length(y_f)/4):end));
    M(f) = abs(A);
end
```

④ From figure g-3 and the zoom in figure g-4, we can figure out that the frequency located at 10, 20, 30… will be emphasized and frequency at 5, 15, 25… will be suppressed. Note that you should take care "transient response" and "steady state response" (blue & red blocks in figure g-1 & g-2) issue when calculating the magnitude response (slide 39~40, Topic2_LTISystems_Part2_HandWriting0407 _2021.pdf). We choose to plot the magnitude response by using the steady state response, which is the desired output when we design the system.

⑤ Figure

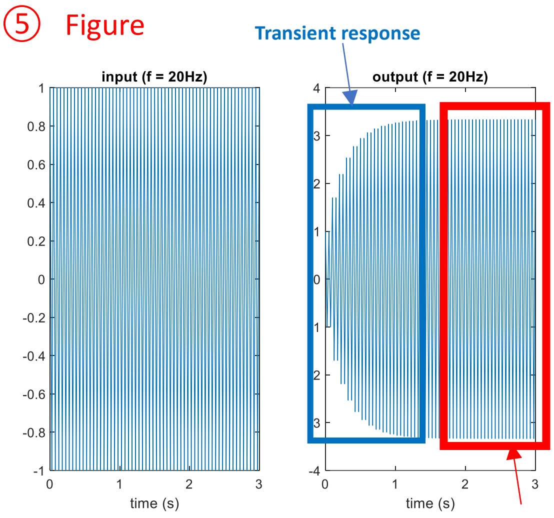


Figure: g-1

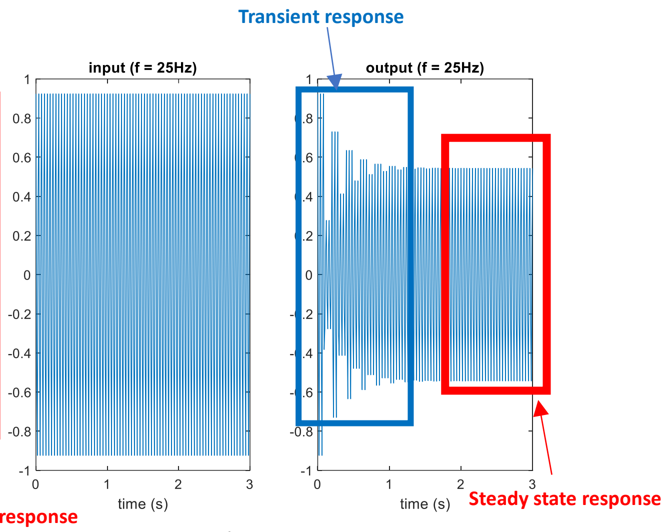


Figure: g-2

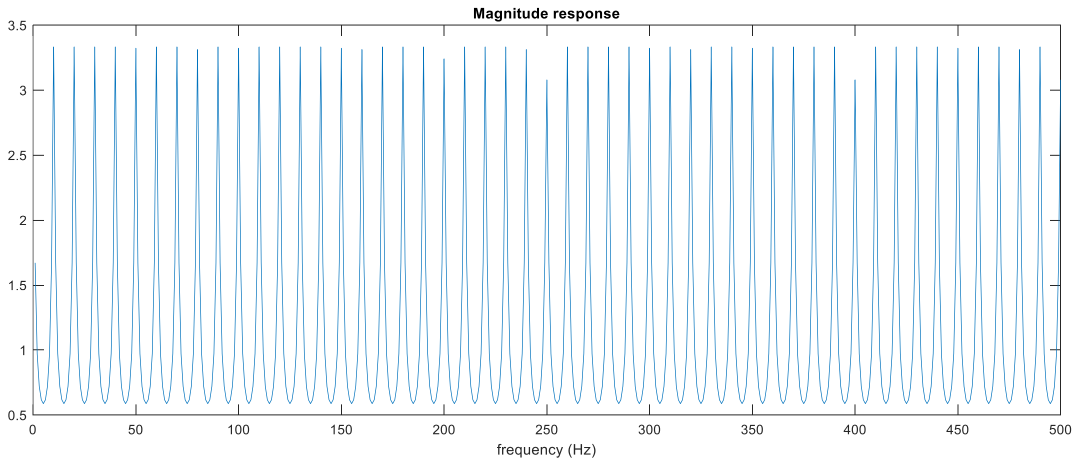


Figure: g-3

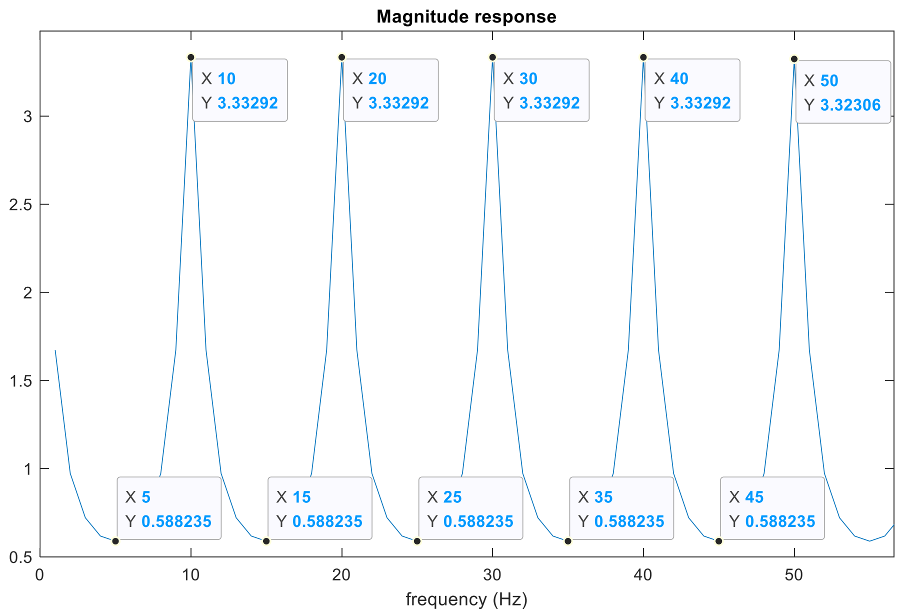


Figure: g-4

Hint:

1. You should plot the magnitude response by the given definition in class. (Should not use ready-to-use MATLAB built in function.)

Bonus (5 pts) Do you know why the magnitude response required in (g) is only plotted for frequency from 0 to Fs/2, instead of from 0 to Fs or even higher frequency, e.g., 2Fs? Please comment it. (Hint: think about what you learn in Topic 1 and computer HW1). (2.5 pts for periodic and symmetric each)

- Because the input is a discrete sinusoidal signal, it is ***periodic in frequency***. Thus, we only need to observe normalized frequency from -0.5~0.5, where 0~0.5 corresponds to absolute frequency 0~Fs/2, and -0.5~0 corresponds to Fs/2~Fs. In other words, we only need to plot 0~Fs with the property of periodic in frequency.
- The cosine signals at positive and negative normalized frequency are the same. Therefore, we only need to plot magnitude response with positive normalized frequency from 0 ~ 0.5, which is equal to 0 ~ Fs/2.