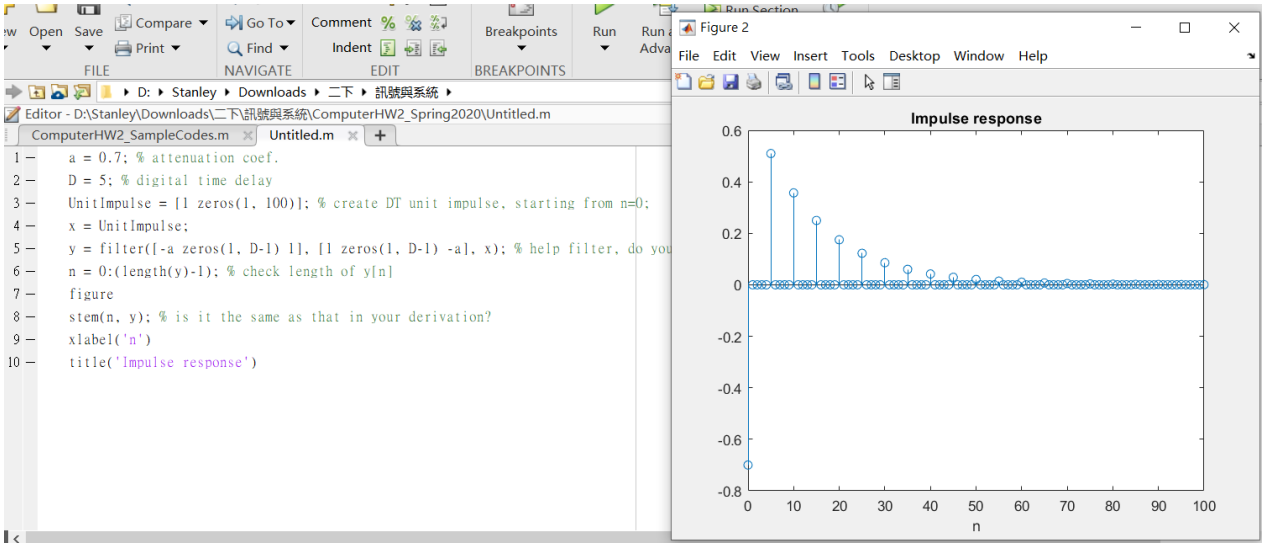
EECS2020 常安彥 107081028 Computer HW2 04/26/2020

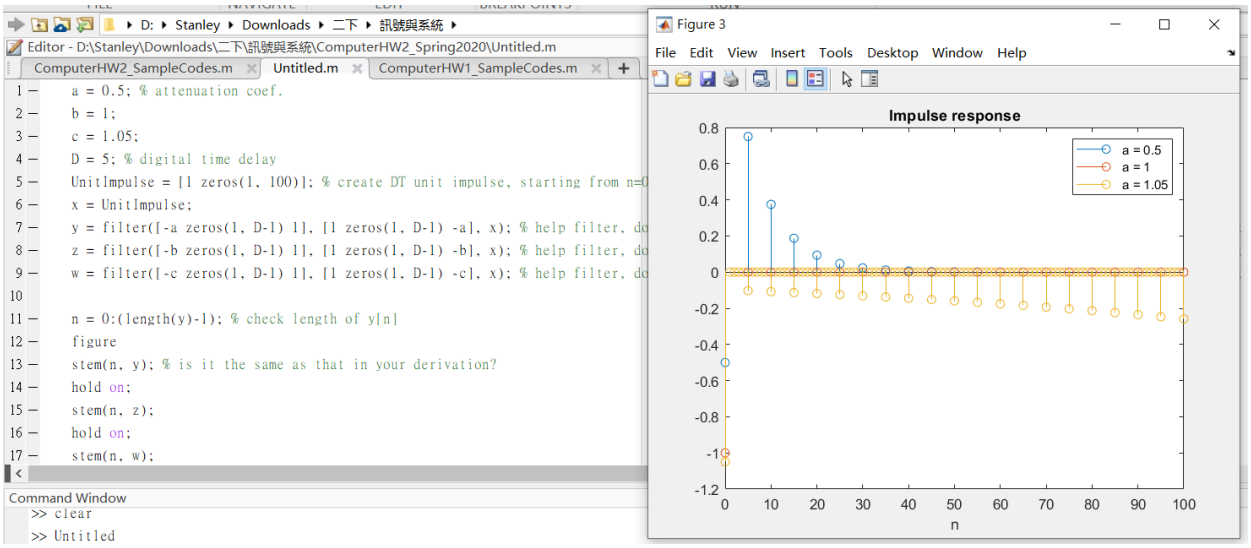
Part 1

(a) Derive the impulse response of this allpass 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.

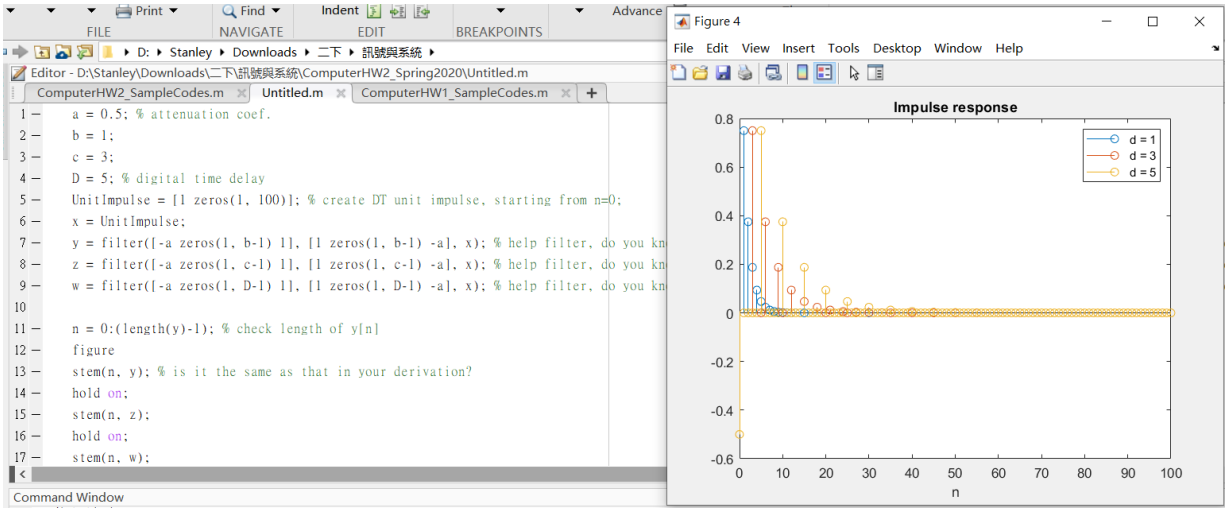


The figure and code are above. y[0] is -0.7 because of -ax[n].

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

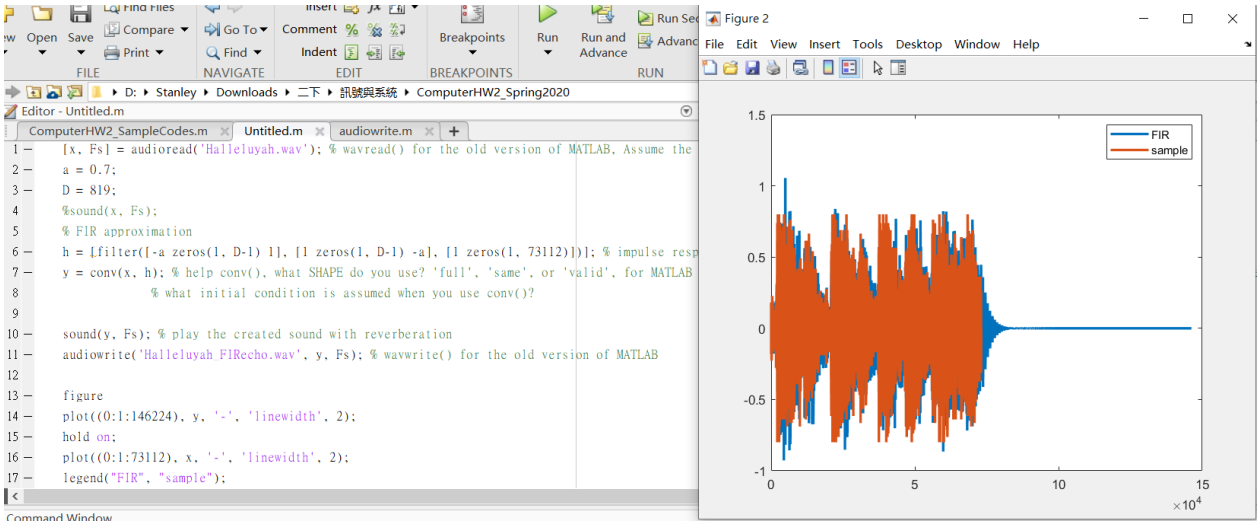


Yes, it can be implemented in real time, because taking preceding inputs is possible. 'a' mustn't be bigger than 1, otherwise the sound reverbing would be larger than before, plotted above. While a = 1, the reverberator is stable because it remains 0 as plotted upwards.



D doesn't matter in this case, not deciding if the reverberator is stable or not. Nevertheless, *D* should be suitable for humans' ear, otherwise the echo decays too rapidly when too small, and reflects too slowly when too big.

(c) Approximate this allpass reverberator by a FIR system

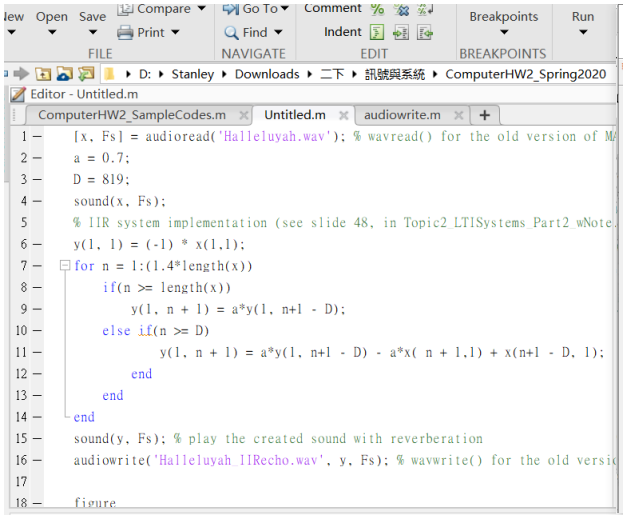


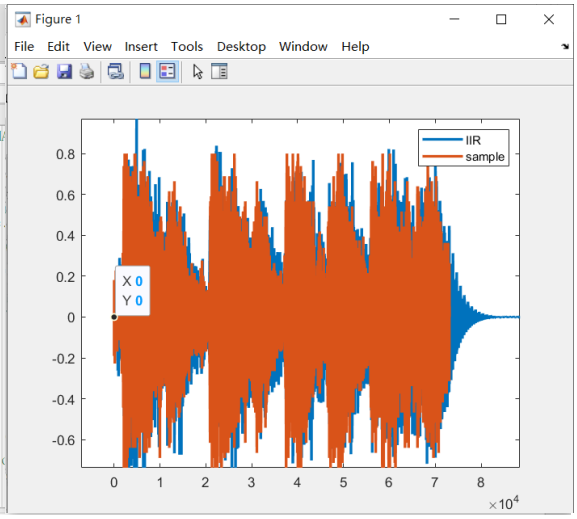
h[n] is the impulse response we got in (a) with *D*(0.1*Fs) read from the audio, while y is the convolution of the sample and h. Here, h[n - k] is the b_k we implement in the FIR form, and D should be an integer. I plotted x and y together, and y is a little bit longer than x, showing the echo.

I consider the domain to set for the convolution the range of x, because the whole sample can be convoluted then. Therefore, x[0] will affect $y[n_{max}]$, which equals $\sum_{k=0}^{nmax} x[k]h[n_{max}-k]$.

The initial condition is that when the index is less than 0, the reverb is 0.

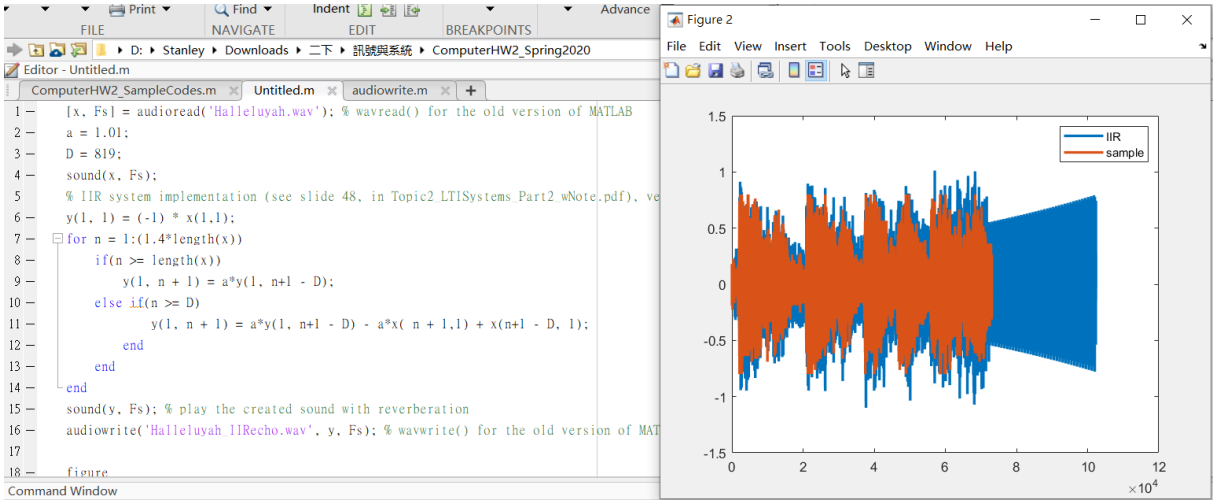
(d) Write your own MATLAB codes to implement this reverberator using the provided recursive LCCDE.





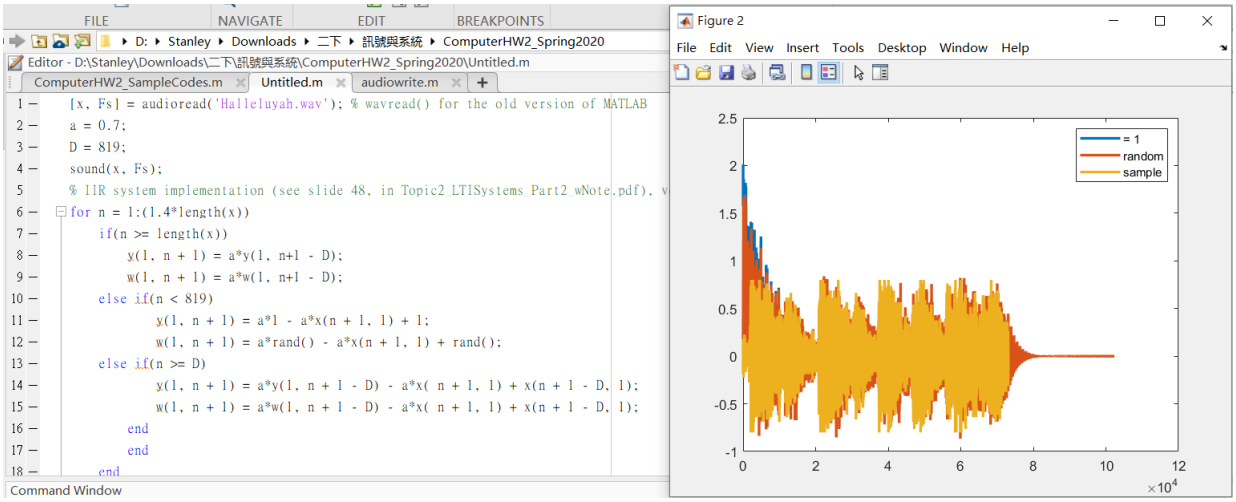
Initial condition is x[0] = 0, and y[0] = 0. X is the unmodified sample, y is modified. (c) and (d) is roughly the same, and the difference is that y in (c) is not affected by the preceding y compared to (d). However, if the range of h[n] covers the range of the sample x, the result will be similar to the method in (d).

(e) 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 is too slow.



I changed *a* from 1 to 1.01, but the result significantly increased. The echo lasted for a long time, and continuously grew louder.

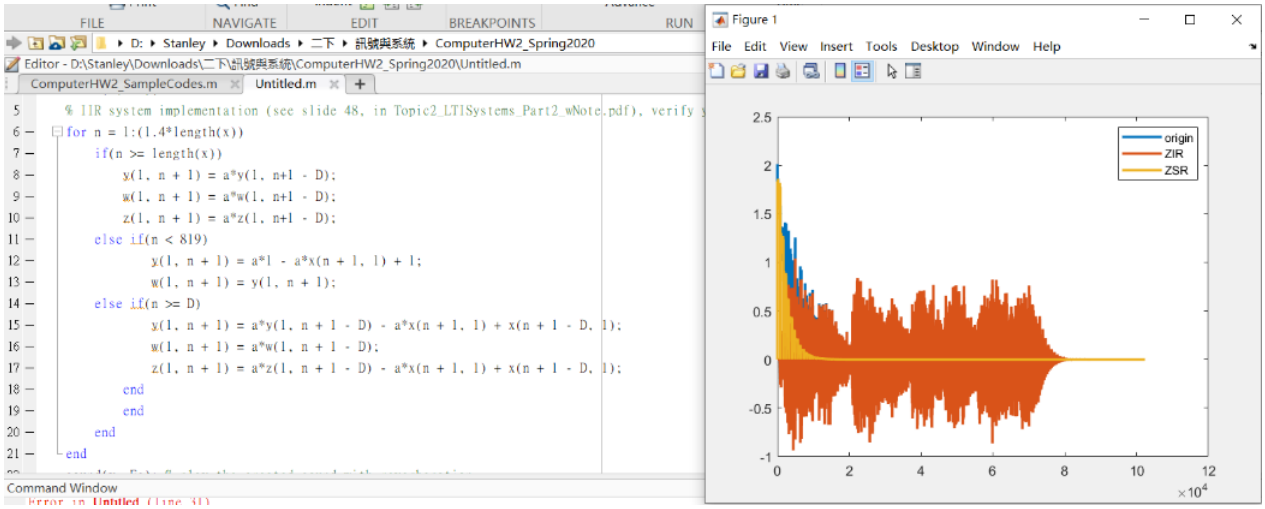
(f) Change the initial condition



Only the beginnings of the sample had changed just like some noise. The following part stuck back to the origin.

The initial condition of the blue plot is that when n < 0, x[n] and y[n] are both 0. On the other hand, the initial condition of the red one is both random.

(g) Following (f), verify whether the system output y[n] is equal to the zero-input response plus zero-state response.



The picture showed that ZIR plus ZSR is very close to the origin response. Array w is ZIR and z is ZSR.

Part 2

Noise remover for echo time estimation

```
%% ---- Echo signal contaminated by noise
y = y_woNoise + randn(1, length(y_woNoise))*0.5; % randn() generate random numbers with zero-mean normal distribution, i.e., Gaussian distribution

figure
plot( (0:(length(y_woNoise)-1))*(1/Fs), y)
xlabel('Time (sec)');
ylabel('Amplitude')
title('Noisy echo signal, Can you find the echo time using your eyes?')

h = [zeros(1, 1300) x zeros(1,1000)]; % impulse response of the noise remover
g = y + fliplr(h);
y_NoiseSuppressed = y + fliplr(g); % noise suppressed by the noise remover, help conv(), what SHAPE do you use? 'full', 'same', or 'valid', for MATLAB 2018 or up. Give SHAPE a try and see the di

Envelope = abs(hilbert(y_NoiseSuppressed));
[PeakValue, EchoTimeIndex] = max(Envelope); % here we use the peak time as the echo time
EchoTime = EchoTimeIndex*(1/Fs) % should be close to the one you got for perfect echo signal
```

```
Command Window
EchoTime =

    17.7100
```

The echo time got for perfect echo signal : 18.0100

```
EchoTime =

    18.0100

fx >>
```

The echo for the suppressed noise : 17.7100

I implemented the fliplr() function and used the ground truth to remove the random noise.