Lab 2: Linear Time-Invariant Systems

Author	Name & Student ID : 王卓扬（12112907）、冯彦捷（12010825）

Introduction

1. Use MATLAB to verify some properties of convolution, including the commutative, associative and distributive properties.

2. Use MATLAB to verify some properties of DT LTI Systems, including linearity, time-invariance.

3. Use MATLAB function *conv* to calculate finite impulse response (FIR), and the outputs of LTI system; Use the function *filter* to calculate infinite impulse response (IIR) of DT LTI Systems described by a constant-coefficient difference equation.

4. Use MATLAB function *load* to load data from hard disk.

5. Use MATLAB to do echo cancellation via inverse filtering.

6. Use MATLAB to calculate and analyze the autocorrelation signal.

7. In order to improve the efficiency, we developed a class named DTS to encapsulate common functions for DT signal processing.

Lab results & Analysis:

■ 2.4 Properties of Discrete-Time LTI Systems

In this exercise, you will verify the commutative, associative and distributive properties of convolution for a specific set of signals. In addition, you will examine the implications of these properties for series and parallel connections of LTI systems. The problems in this exercise will assume that you are comfortable and familiar with the conv function described in Tutorial 2.1. Although the problems in this exercise solely explore discrete-time systems, the same properties are also valid for continuous-time systems.

Basic Problems

(a). Many of the problems in this exercise will use the following three signals:

$$x_1[n] = \begin{cases} 1, & 0 \le n \le 4, \\ 0, & \text{otherwise}, \end{cases}$$

$$h_1[n] = \begin{cases} 1, & n = 0, \\ -1, & n = 1, \\ 3, & n = 2, \\ 1, & n = 4, \\ 0, & \text{otherwise}, \end{cases}$$

$$h_2[n] = \begin{cases} 2, & n = 1, \\ 5, & n = 2, \\ 4, & n = 3, \\ -1, & n = 4, \\ 0, & \text{otherwise}. \end{cases}$$

Define the MATLAB vector `x1` to represent $x_1[n]$ on the interval $0 \leq n \leq 9$, and the vectors `h1` and `h2` to represent $h_1[n]$ and $h_2[n]$ for $0 \leq n \leq 4$. Also define `nx1` and `nh1` to be appropriate index vectors for these signals. Make appropriately labeled plots of all the signals using `stem`.

Solution (a):

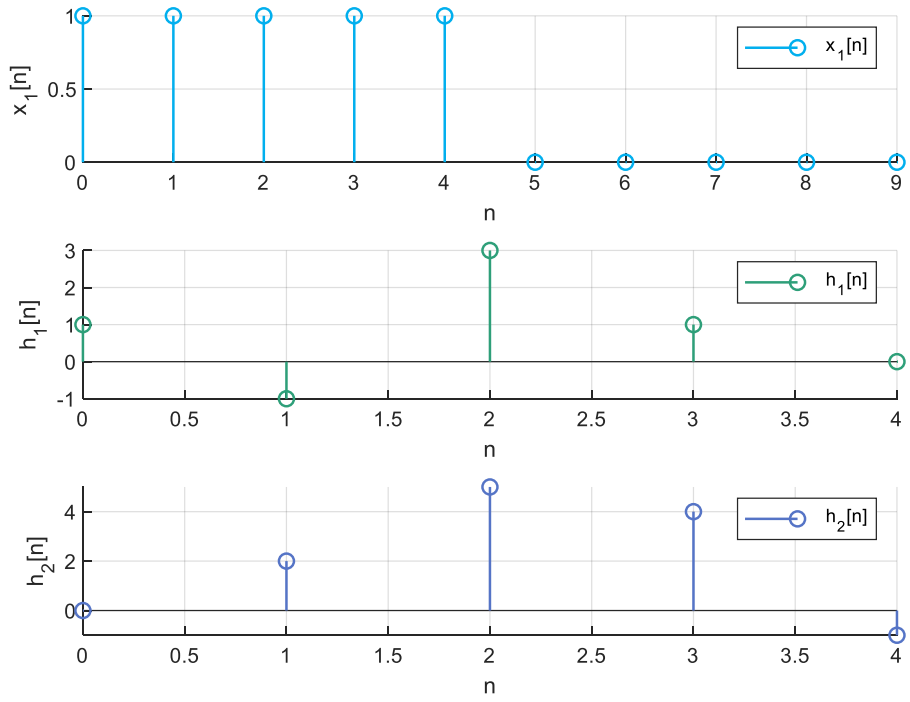


Fig 2.4.a

Analysis (a): Fig 2.4.a shows the patterns of x1[n], h1[n] and h2[n] which are defined in the problem (a).

(b). The commutative property states that the result of a convolution is the same regardless of the order of the operands. This implies that the output of an LTI system with impulse response $h[n]$ and input $x[n]$ will be the same as the output of an LTI system with impulse response $x[n]$ and input $h[n]$. Use `conv` with `h1` and `x1` to verify this property. Is the output of `conv` the same regardless of the order of the input arguments?

Solution (b):

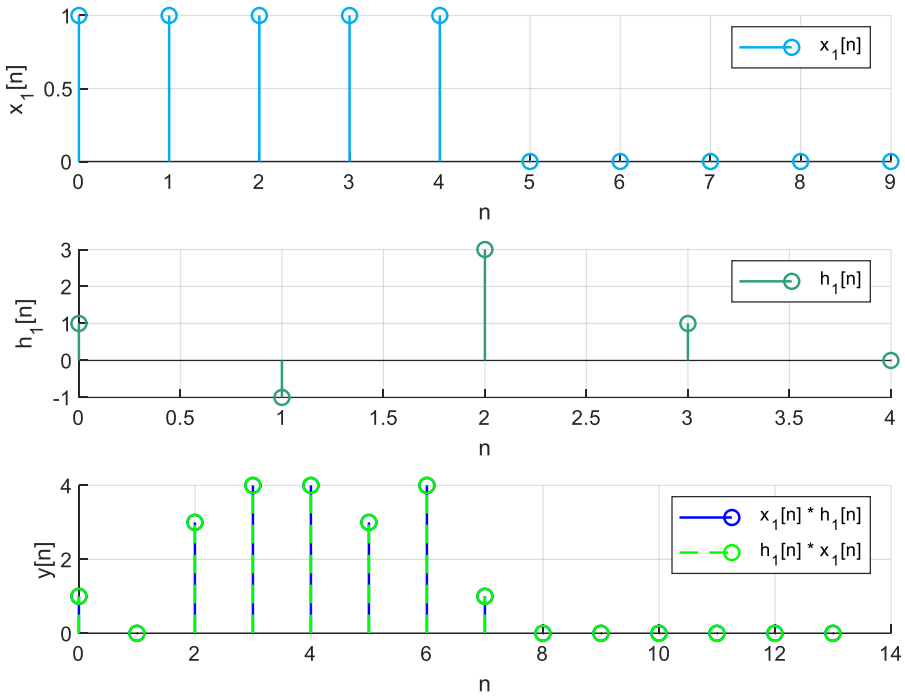


Fig 2.4.b

Analysis (b): We plot the two signals y1[n] = x1[n] * h1[n] and y2[n] = h1[n] * x1[n], and we can see that y1[n] = y2[n] in Fig 2.4.b. That verifies the commutative property.

(c). Convolution is also distributive. This means that

$$x[n] * (h_1[n] + h_2[n]) = x[n] * h_1[n] + x[n] * h_2[n] .$$

This implies that the output of two LTI systems connected in parallel is the same as one system whose impulse response is the sum of the impulse responses of the parallel systems. Figure 2.8 illustrates this property.

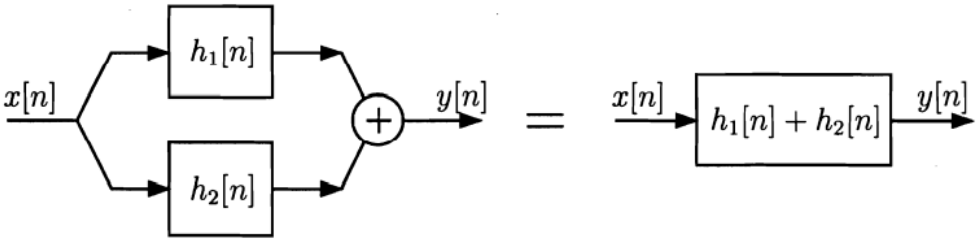


Figure 2.8. Distributive property of convolution.

Verify the distributive property using `x1`, `h1` and `h2`. Compute the sum of the outputs of LTI systems with impulse responses $h_1[n]$ and $h_2[n]$ when $x_1[n]$ is the input. Compare this with the output of the LTI system whose impulse response is $h_1[n] + h_2[n]$ when the input is $x_1[n]$. Do these two methods of computing the output give the same result?

Solution (c):

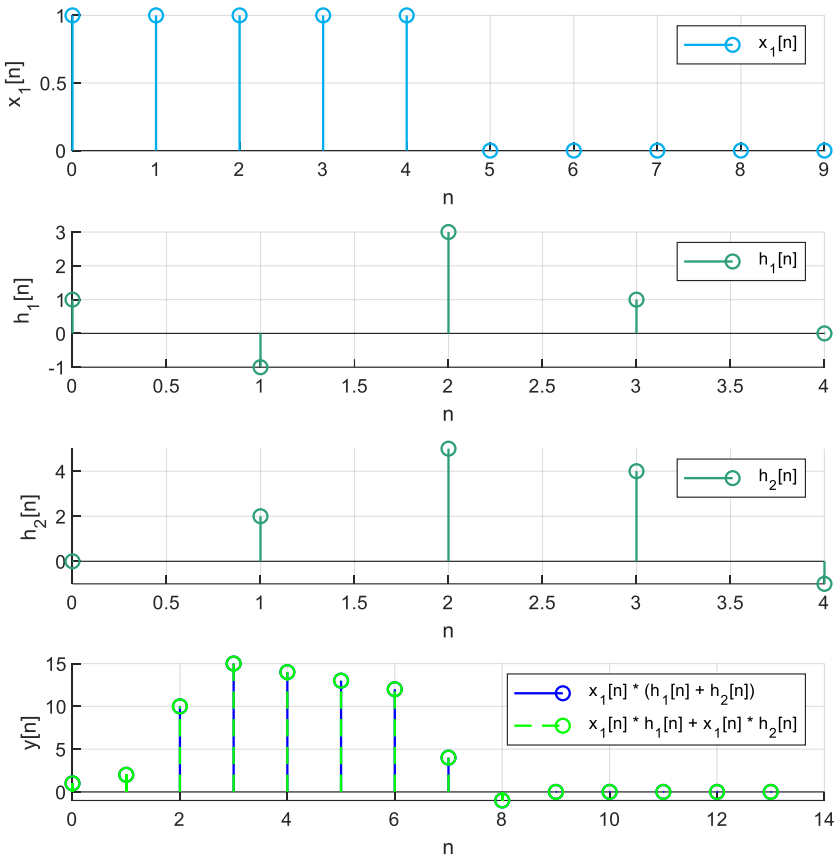


Fig 2.4.c

Analysis (c): In Fig 2.4.c we see that x1[n] * (h1[n] + h2[n]) = x1[n] * h1[n] + x1[n] * h2[n]. That verifies the distributive property.

(d). Convolution also possesses the associative property, i.e.,

$$(x[n] * h_1[n]) * h_2[n] = x[n] * (h_1[n] * h_2[n]) .$$

This property implies that the result of processing a signal with a series of LTI systems is equivalent to processing the signal with a single LTI system whose impulse response is the convolution of all the individual impulse responses of the connected systems. Figure 2.9 illustrates this property for the case of two LTI systems connected in series.

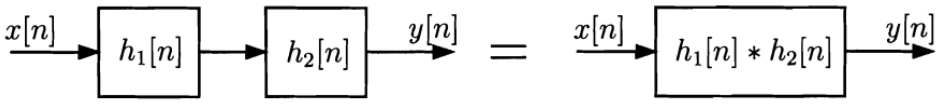


Figure 2.9. Associative property of convolution.

Use the following steps to verify the associative property using **x1**, **h1** and **h2**:

- Let $w[n]$ be the output of the LTI system with impulse response $h_1[n]$ shown in Figure 2.9. Compute $w[n]$ by convolving $x_1[n]$ and $h_1[n]$.
- Compute the output $y_{d1}[n]$ of the whole system by convolving $w[n]$ with $h_2[n]$.

- Find the impulse response $h_{\text{series}}[n] = h_1[n] * h_2[n]$.
- Convolve $x_1[n]$ with $h_{\text{series}}[n]$ to get the output $y_{d2}[n]$.

Compare $y_{d1}[n]$ and $y_{d2}[n]$. Did you get the same results when you process $x_1[n]$ with the individual impulse responses as when you process it with $h_{\text{series}}[n]$?

Solution (d):

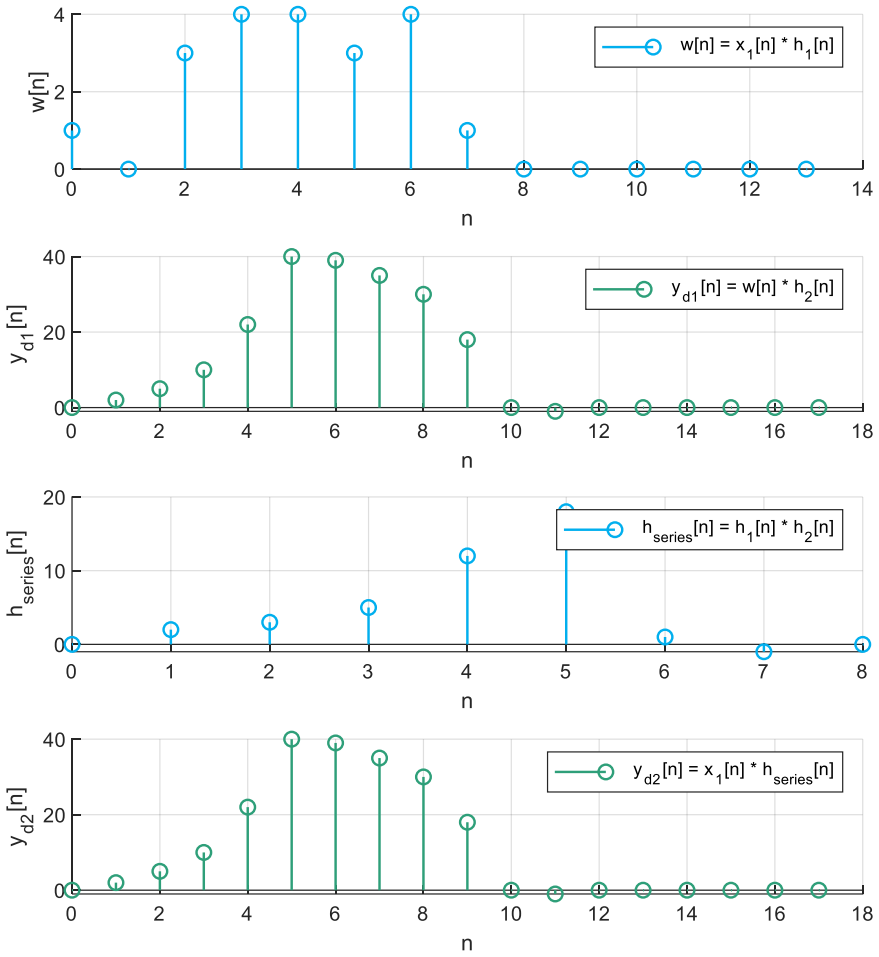


Fig 2.4.d

Analysis (d): From Fig 2.4.d we can find that yd1[n] = yd2[n], that is (x1[n] * h1[n]) * h2[n] = x1[n] * (h1[n] * h2[n]). That verifies the associative property.

Intermediate Problems

(e). Suppose two LTI systems have impulse responses $h_{e1} = h_1[n]$ and $h_{e2}[n] = h_1[n - n_0]$, where $h_1[n]$ is the same signal defined in Part (a) and n_0 is an integer. Let $y_{e1}[n]$ and $y_{e2}[n]$ be the outputs of these systems when $x[n]$ is the input. Use the commutative property to argue that the outputs will be the same if you interchange the input and impulse response of each system. Notice that once you have done this the two systems have the same impulse response and the inputs are delayed versions of the same signal. Based on this observation and time-invariance, argue that $y_{e2}[n] = y_{e1}[n - n_0]$. Use MATLAB to confirm your answer for the case when $n_0 = 2$ and the input $x[n]$ is the signal $x_1[n]$ defined in Part (a).

Solution (e):

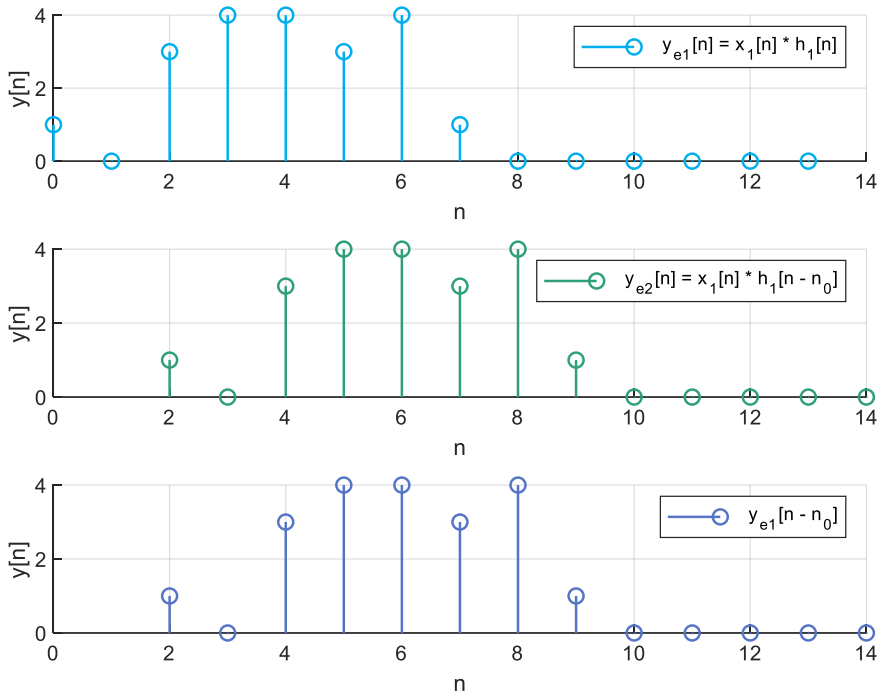


Fig 2.4.e

Analysis (e):

(1) According to the commutative property, y[n] = x[n] * h[n] = h[n] * x[n], so the outputs will be the same if interchanging the input and impulse response of each system;

(2) Applying the properties we have ye2[n] = x[n] * h1[n – n0] = h1[n – n0] * x[n], and ye1[n] = x[n] * h1[n] = h1[n] * x[n], so ye1[n – n0] = h1[n – n0] * x[n], then we have ye2[n] = ye1[n – n0];

(3) From Fig 2.4.e, we can see ye2[n] = ye1[n – 2]. The answer is confirmed.

(f). Consider two systems connected in series; call them System 1 and System 2. Suppose System 1 is a memoryless system and is characterized by the input/output relationship $y[n] = (n+1)x[n]$, and System 2 is LTI with impulse response $h_{f2}[n] = h_1[n]$ as defined in Part (a). Suppose you decide to investigate whether or not the associative property of convolution holds for the series connection of these two systems by following the steps:

- Let $w[n]$ be the output of System 1 when the input is $x_1[n]$ as defined above. Use `nx1` and `x1` with the termwise multiplication operator `.*` to define a MATLAB vector `w` to represent $w[n]$.
- Use $w[n]$ as the input to System 2, and let the output of that system be $y_{f1}[n]$. Compute `yf1` in MATLAB using `w` and `h1`.
- Let $h_{f1}[n]$ be the output of System 1 when the input to the system is $\delta[n]$. Define a vector `hf1` to represent this signal over the interval $0 \leq n \leq 4$.
- Let $h_{\text{series}}[n] = h_{f1}[n] * h_{f2}[n]$. Compute a vector `hseries` to represent this signal.
- Let $y_{f2}[n]$ be the output of an LTI system whose impulse response is $h_{\text{series}}[n]$ when the input is $x_1[n]$. Compute `yf2` in MATLAB using `hseries` and `x1`.

Does $y_{f1}[n] = y_{f2}[n]$? If so, why would you expect it to? If not, this means the result of processing a signal with the series connection of Systems 1 and 2 is not equal to the result of processing the signal with a single system whose impulse response is the convolution of the impulse response of System 1 with the impulse response of System 2. Does this violate the associative property of convolution as discussed in Part (d)?

Solution (f):

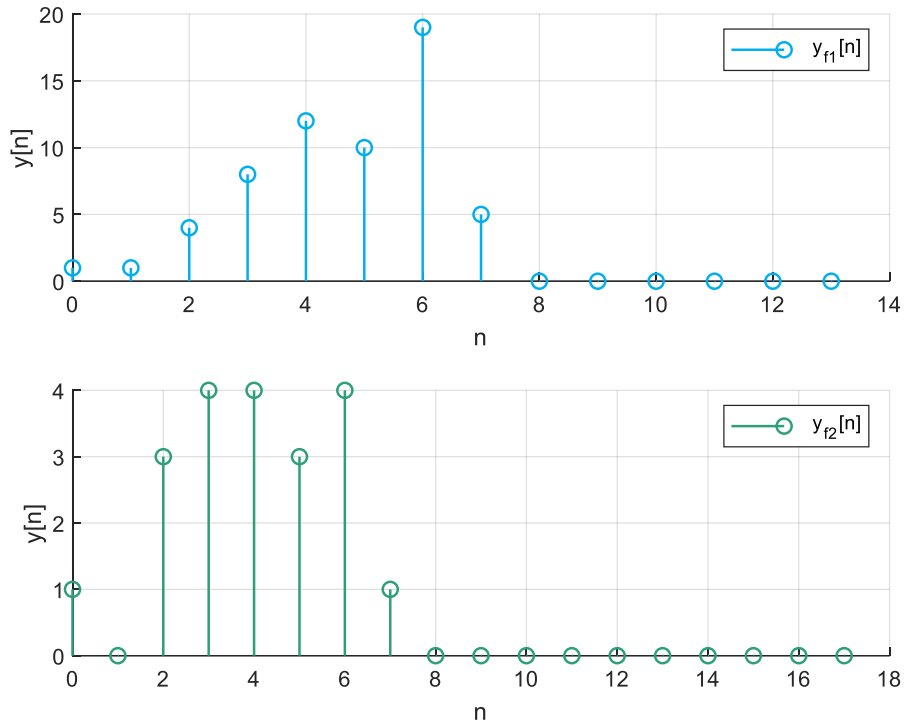


Fig 2.4.f

Analysis (f): From Fig 2.4.f we can see that yf1[n] ~= yf2[n]. But it doesn't violate the associative property, because only for LTI systems the outputs y[n] can be represented as x[n] convoluting with h[n]. However, in (f) the System 1 is not a LTI system, which does not meet the fundamental conditions.

(g). Consider the parallel connection of two systems; call them System 1 and System 2. System 1 is a memoryless system characterized by the input/output relationship $y[n] = x^2[n]$. System 2 is an LTI system with impulse response $h_{g2}[n] = h_2[n]$ as defined in Part (a). Suppose you were to use the steps below to investigate whether or not the distributive property of convolution held for the parallel connection of these systems:

- Let $y_{ga}[n]$ be the output of System 1 when the input is the signal $x_g[n] = 2\delta[n]$. Define `xg` to represent this input over the interval $0 \leq n \leq 4$, and use `xg` and the termwise exponentiation operator `.^` to define a MATLAB vector `yga` to represent $y_{ga}[n]$.
- Let $y_{gb}[n]$ be the output of System 2 when $x_g[n]$ is the input, and define `ygb` to represent this signal.
- Let $y_{g1}[n]$ be the sum of $y_{ga}[n]$ and $y_{gb}[n]$, the outputs of the parallel branches. Define the vector `yg1` to represent $y_{g1}[n]$. Note that because `yga` is shorter in length than `ygb`, you will have to extend `yga` with some zeros before you can add the vectors.

- Let $h_{g1}[n]$ be the output of System 1 when the input is $\delta[n]$. Define `hg1` to represent this signal on the interval $0 \leq n \leq 4$.
- Let $h_{\text{parallel}}[n]$ be $h_{g1}[n] + h_{g2}[n]$. Define `hparallel` to represent this signal.
- Let $y_{g2}[n]$ be the output of the LTI system with impulse response $h_{\text{parallel}}[n]$ when the input is $x_g[n]$. Define a vector `yg2` to represent this signal.

Are `yg1` and `yg2` equal? If so, why would you expect this? If not has the distributive property of convolution been violated ?

Solution (g):

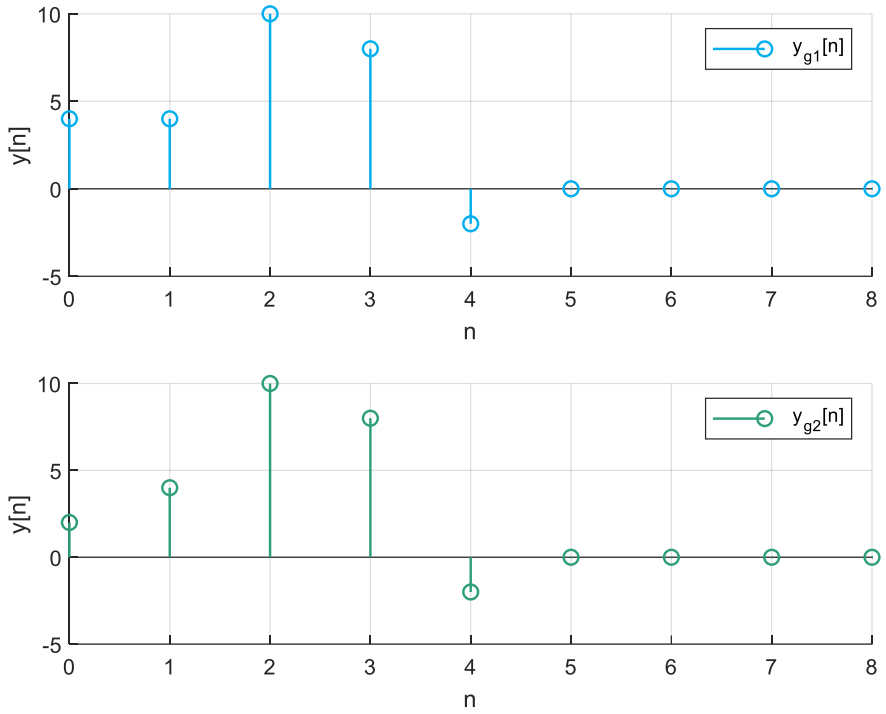


Fig 2.4.g

Analysis (g): From Fig 2.4.g we can see that yg1[n] ~= yg2[n] on n = 0. But it doesn't violate the distributive property, because only for LTI systems the outputs y[n] can be represented as x[n] convoluting with h[n]. However, in (g) the System 1 is not a LTI system, which does not meet the fundamental conditions.

■ 2.5 Linearity and Time-Invariance

In this exercise you will become more familiar with the system properties of linearity and time-invariance. In particular, you will be presented with a number of systems, and then be asked to determine if they are linear or time-invariant. This exercise also explores an important property of discrete-time LTI systems: if $h[n]$ is the response to the unit impulse $\delta[n]$, then the response of the system $y[n]$ to any input $x[n]$ is determined by the convolution sum, Eq. (2.3). An analogous property holds for continuous-time LTI systems.

The problems in this exercise assume that you are familiar with the functions `conv` and `filter`. These functions are explained in Tutorial 2.1 and Tutorial 2.2.

Basic Problems

Consider the following three systems:

$$\begin{aligned} &\text{System 1:} && w[n] = x[n] - x[n-1] - x[n-2]\,, \\ &\text{System 2:} && y[n] = \cos(x[n])\,, \\ &\text{System 3:} && z[n] = n\,x[n]\,, \end{aligned}$$

where $x[n]$ is the input to each system, and $w[n]$, $y[n]$, and $z[n]$ are the corresponding outputs.

(a). Consider the three inputs signals $x_1[n] = \delta[n]$, $x_2[n] = \delta[n-1]$, and $x_3[n] = \big(\delta[n] + 2\,\delta[n-1]\big)$. For System 1, store in `w1`, `w2`, and `w3` the responses to the three inputs. The vectors `w1`, `w2`, and `w3` need to contain the values of $w[n]$ only on the interval $0 \le n \le 5$. Use `subplot` and `stem` to plot the four functions represented by `w1`, `w2`, `w3`, and `w1+2*w2` within a single figure. Make analogous plots for Systems 2 and 3.

Solution (a):

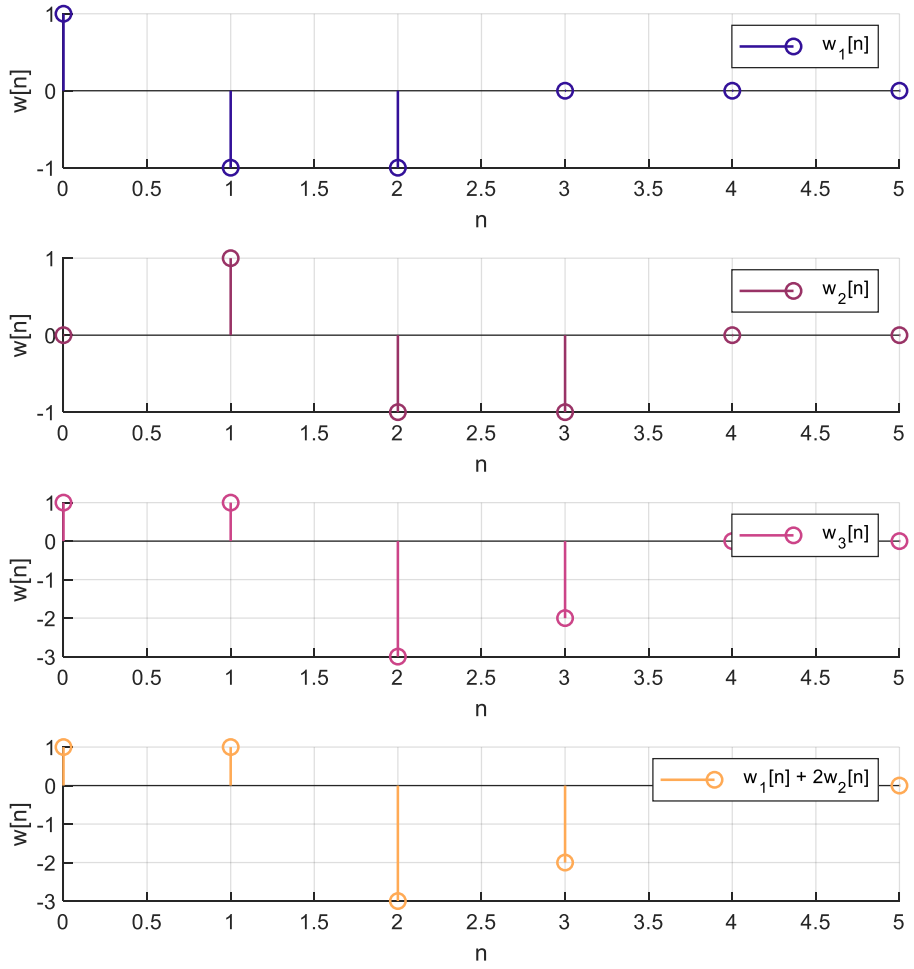


Fig 2.5.a (1)

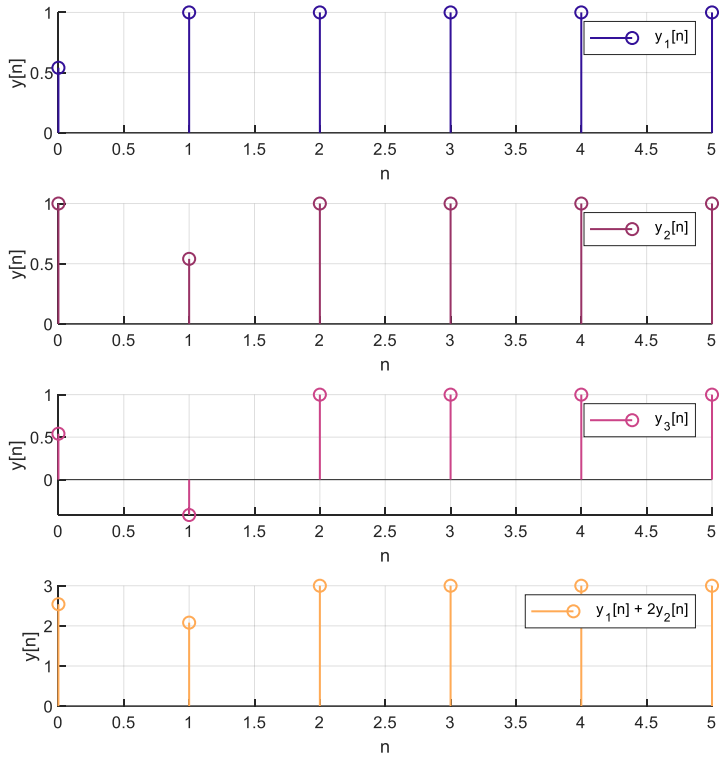


Fig 2.5.a (2)

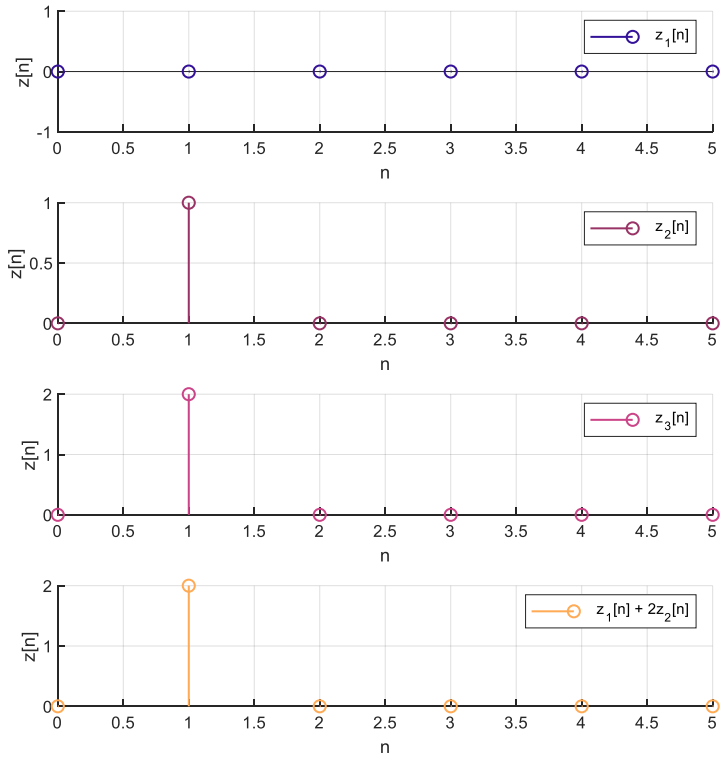


Fig 2.5.a (3)

Analysis (a): Fig 2.5.a (1) ~ (3) show the four relevant signals' patterns for w, y and z.

(b). State whether or not each system is linear. If it is linear, justify your answer. If it is not linear, use the signals plotted in Part (a) to supply a counter-example.

Solution (b):

Conclusion:

System 1 is **linear**;

System 2 is **non-linear**;

System 3 is **linear**.

Analysis (b):

(1) Let

$$x_1[n] \to w_1[n] = x_1[n] - x_1[n-1] - x_1[n-2]$$
$$x_2[n] \to w_2[n] = x_2[n] - x_2[n-1] - x_2[n-2]$$

Consider

$$Ax_1[n] + Bx_2[n] \to w[n]$$

And we have

$$\begin{aligned} w[n] &= (Ax_1[n] + Bx_2[n]) - (Ax_1[n-1] + Bx_2[n-1]) - (Ax_1[n-2] + Bx_2) \\ &= A(x_1[n] - x_1[n-1] - x_1[n-2]) + B(x_2[n] - x_2[n-1] - x_2[n-2]) \\ &= Aw_1[n] + Bw_2[n] \end{aligned}$$

That is

$$Ax_1[n] + Bx_2[n] \to Aw_1[n] + Bw_2[n]$$

So System 1 is **linear**;

(2) As the Fig 2.5.a (2) shown

$$y_3[n] \neq y_1[n] + 2 \times y_2[n]$$

which means System 2 is **non-linear**;

(3) Let

$$x_1[n] \to z_1[n] = n \times x_1[n]$$
$$x_2[n] \to z_2[n] = n \times x_2[n]$$

Consider

$$Ax_1[n] + Bx_2[n] \to z[n]$$

And we have

$$\begin{aligned} z[n] &= n \times (Ax_1[n] + Bx_2[n]) \\ &= A(n \times x_1[n]) + B(n \times x_2[n]) \\ &= Az_1[n] + Bz_2[n] \end{aligned}$$

That is

$$Ax_1[n] + Bx_2[n] \to Az_1[n] + Bz_2[n]$$

So System 3 is **linear**.

(c). State whether or not each system is time-invariant. If it is time-invariant, justify your answer. If it is not time-invariant, use the signals plotted in Part (a) to supply a counter-example.

Solution (c):

System 1 is **time-invariant**;

System 2 is **time-invariant**;

System 3 is **time-varying**.

Analysis (c):

(1) Let

$$x[n] \rightarrow w[n] = x[n] - x[n-1] - x[n-2]$$

Consider

$$x[n-n_0] \rightarrow w'[n]$$

And we have

$$w'[n] = x[n-n_0] - x[n-n_0-1] - x[n-n_0-2]$$
$$= w[n-n_0]$$

So System 1 is **time-invariant**.

(2) Let

$$x[n] \rightarrow y[n] = \cos(x[n])$$

Consider

$$x[n-n_0] \rightarrow y'[n]$$

And we have

$$y'[n] = \cos(x[n-n_0]) = y[n-n_0]$$

That means System 2 is **time-variant**.

(3) As the Fig 2.5.a (3) shown, when

$$x_1[n] \rightarrow z_1[n]$$
$$x_1[n-1] \rightarrow z_2[n]$$

We find that

$$z_2[n] \neq z_1[n-1]$$

So System 3 is **time-varying**.

Intermediate Problems

In these problems, you will be asked to consider how the impulse response can be used to calculate the step response of an LTI system. Consider the two causal systems defined by the following linear difference equations:

$$\text{System 1:} \qquad y_1[n] = (3/5)\, y_1[n-1] + x[n]\,,$$
$$\text{System 2:} \qquad y_2[n] = (3/5)^n\, y_2[n-1] + x[n]\,,$$

where each system satisfies initial rest conditions. Initial rest conditions state that if $x[n] = 0$ for $n \leq n_0$, then $y[n] = 0$ for $n \leq n_0$. Define $h_1[n]$ and $h_2[n]$ to be the responses of Systems 1 and 2, respectively, to the signal $\delta[n]$.

(d). Calculate $h_1[n]$ and $h_2[n]$ on the interval $0 \leq n \leq 19$, and store these responses in `h1` and `h2`. Plot each response using `stem`. Hint: The `filter` function can be used to calculate `h1`. However, System 2 is described by a difference equation with non-constant coefficients; therefore, you must either determine `h2` analytically or use a `for` loop rather than `filter` to calculate `h2`.

Solution (d):

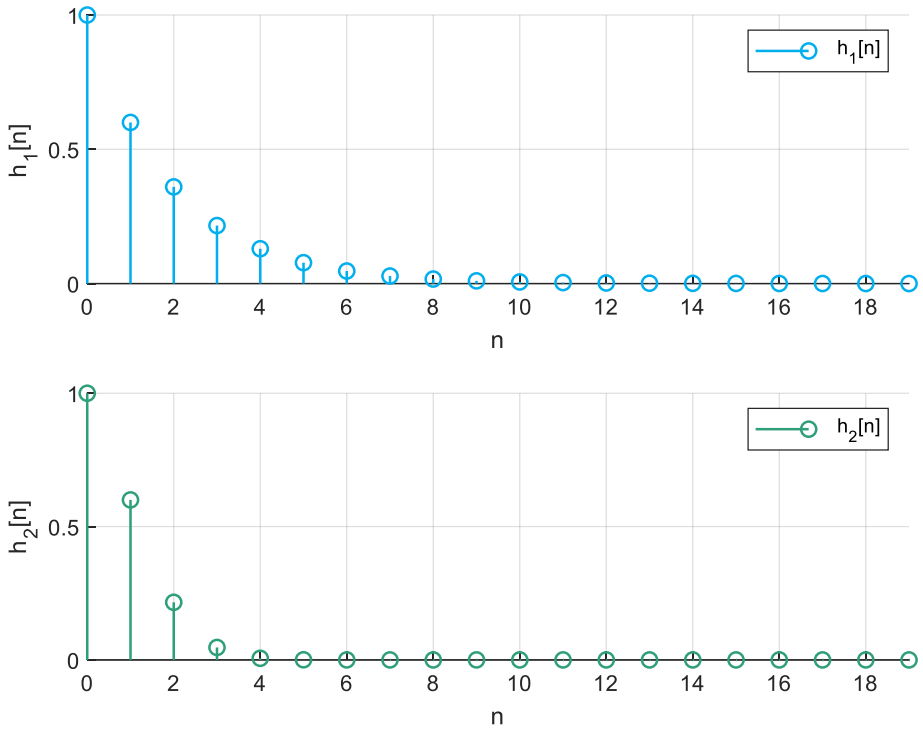


Fig 2.5.d

Analysis (d):

(1) In Fig 2.5.d (top) shows h1[n] which is calculated by using the function *filter*.

(2) In Fig 2.5.d (bottom) shows h2[n] which is calculated analytically. Firstly

$$y_2[n] = 0 \ (\because To\ n < 0, x[n] = 0)$$

And the initial condition is

$$y_2[0] = \left(\frac{3}{5}\right)^1 \times y_2[-1] + x[0] = 1$$

After that, we have $x[n] = 0\ (n > 0)$, so

$$y_2[n] = \left(\frac{3}{5}\right)^n \times y_2[n-1]$$

Calculate the series we obtain

$$y_2[n] = \left(\frac{3}{5}\right)^{1+2+\cdots+n} = \left(\frac{3}{5}\right)^{\frac{n(n+1)}{2}}$$

(e). For each system, calculate the unit step response on the interval $0 \leq n \leq 19$, and store the responses in `s1` and `s2`. Again, `filter` can be used only to calculate the step response of System 1. Use a `for` loop to calculate `s2`.

Solution (e):

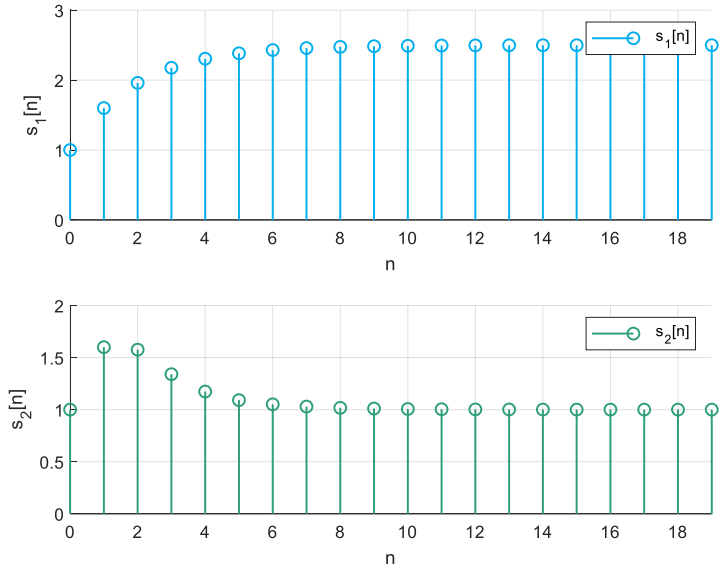


Fig 2.5.e

Analysis (e):

(1) In Fig 2.5.e (top) shows s1[n] which is calculated by using the function *filter*.

(2) In Fig 2.5.e (bottom) shows s2[n] which is calculated with a *for* loop.

(f). Note that $h_1[n]$ and $h_2[n]$ are zero for $n \geq 20$ for all practical purposes. Thus `h1` and `h2` contain all we need to know about the response of each system to the unit impulse. Define $z_1[n] = h_1[n] * u[n]$ and $z_2[n] = h_2[n] * u[n]$, where $u[n]$ is the unit step function. Use `conv` to calculate $z_1[n]$ and $z_2[n]$ on the interval $0 \leq n \leq 19$, and store these calculations in the vectors `z1` and `z2`. You must first define a vector containing $u[n]$ over an appropriate interval, and then select the subset of the samples produced by `conv(h1,u)` and `conv(h2,u)` that represent the interval $0 \leq n \leq 19$. Since you have truncated two infinite-length signals, only a portion of the outputs of `conv` will contain valid sequence values. This issue was also discussed in Exercise 2.7 Part (c).

Solution (f):

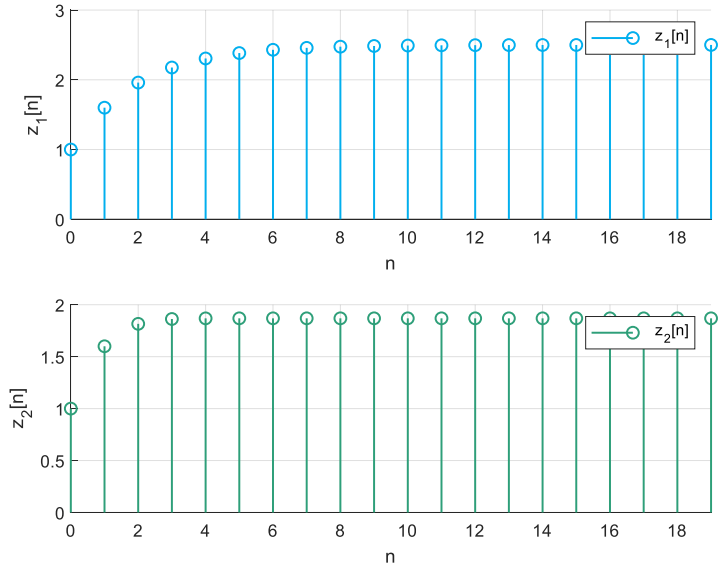


Fig 2.5.f

Analysis (f): z1[n] and z2[n] are plotted in Fig 2.5.f.

(g). Plot **s1** and **z1** on the same set of axes. If the two signals are identical, explain why you could have anticipated this similarity. Otherwise, explain any differences between the two signals. On a different set of axes, plot **s2** and **z2**. Again, explain how you might have anticipated any differences or similarities between these two signals.

Solution (g):

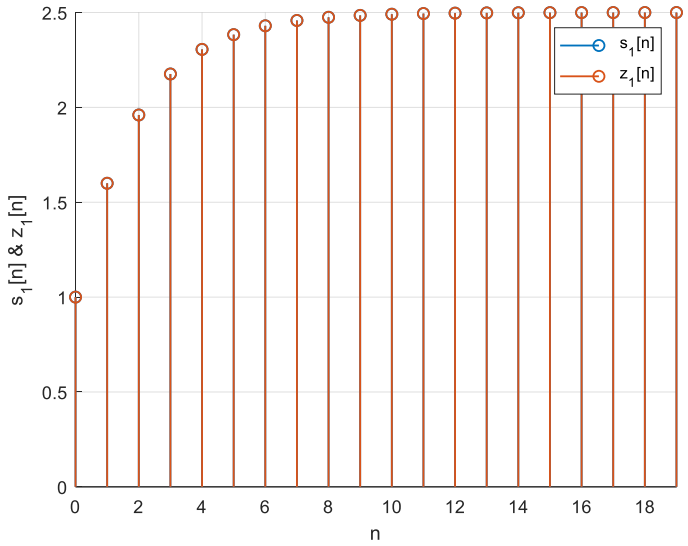


Fig 2.5.g (1)

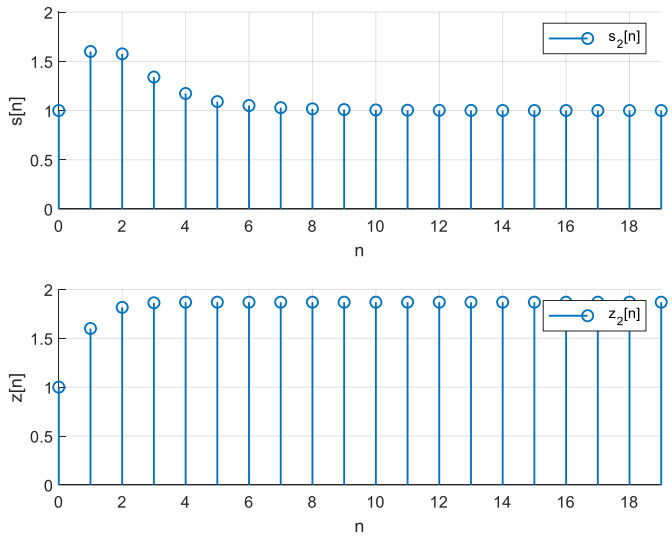


Fig 2.5.g (2)

Analysis (g):

(1) In Fig 2.5.g (1) we see s1[n] = z1[n], because System 1 is an LTI system, which guarantees that s1[n] = u[n] * h1[n] = z1[n].

(2) In Fig 2.5.g (2) we see s2[n] and z2[n] start from the same point but converge to different values, because System 2 is not an LTI system and the convolution property is not guaranteed.

■ 2.10 Echo Cancellation via Inverse Filtering

In this exercise, you will consider the problem of removing an echo from a recording of a speech signal. This project will use the audio capabilities of MATLAB to play recordings of both the original speech and the result of your processing. To begin this exercise you will need to load the speech file `lineup.mat`, which is contained in the Computer Explorations Toolbox. The Computer Explorations Toolbox can be obtained from The MathWorks at the address provided in the Preface. If this speech file is already somewhere in your MATLABPATH, then you can load the data into MATLAB by typing

```
>> load lineup.mat
```

You can check your MATLABPATH, which is a list of all the directories which are currently accessible by MATLAB, by typing `path`. The file `lineup.mat` must be in one of these directories.

Once you have loaded the data into MATLAB, the speech waveform will be stored in the variable `y`. Since the speech was recorded with a sampling rate of 8192 Hz, you can hear the speech by typing

```
>> sound(y,8192)
```

You should hear the phrase "line up" with an echo. The signal $y[n]$, represented by the vector `y`, is of the form

$$y[n] = x[n] + \alpha x[n - N], \tag{2.21}$$

where $x[n]$ is the uncorrupted speech signal, which has been delayed by N samples and added back in with its amplitude decreased by $\alpha < 1$. This is a reasonable model for an echo resulting from the signal reflecting off of an absorbing surface like a wall. If a microphone is placed in the center of a room, and a person is speaking at one end of the room, the recording will contain the speech which travels directly to the microphone, as well as an echo which traveled across the room, reflected off of the far wall, and then into the microphone. Since the echo must travel further, it will be delayed in time. Also, since the speech is partially absorbed by the wall, it will be decreased in amplitude. For simplicity ignore any further reflections or other sources of echo.

For the problems in this exercise, you will use the value of the echo time, $N = 1000$, and the echo amplitude, $\alpha = 0.5$.

Basic Problems

(a). In this exercise you will remove the echo by linear filtering. Since the echo can be represented by a linear system of the form Eq. (2.21), determine and plot the impulse response of the echo system Eq. (2.21). Store this impulse response in the vector **he** for $0 \leq n \leq 1000$.

Solution (a):

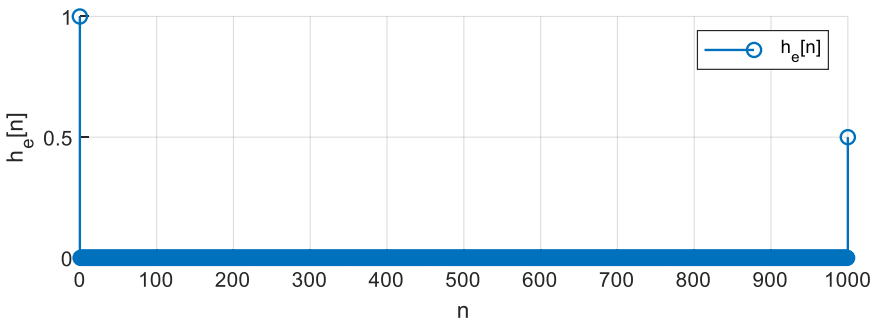


Fig 2.10.a

Analysis (a): Shown in Fig 2.10.a is the signal represented as

$$h_e[n] = \delta[n] + 0.5 \times \delta[n - 1000]$$

(b). Consider an echo removal system described by the difference equation

$$z[n] + \alpha z[n - N] = y[n], \tag{2.22}$$

where $y[n]$ is the input and $z[n]$ is the output which has the echo removed. Show that Eq. (2.22) is indeed an inverse of Eq. (2.21) by deriving the overall difference equation relating $z[n]$ to $x[n]$. Is $z[n] = x[n]$ a valid solution to the overall difference equation?

Solution (b):

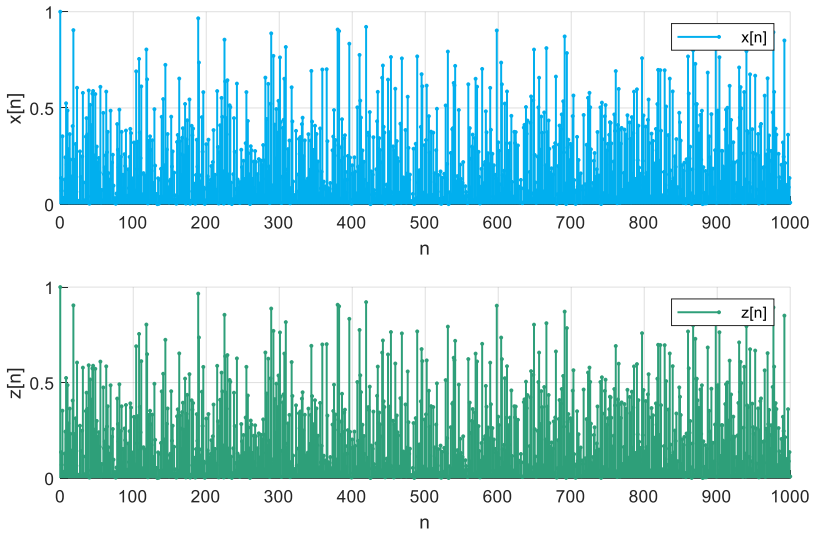


Fig 2.10.b (A random test)

Analysis (b):

(1) We have

$$y[n] = x[n] + \alpha \cdot x[n - N]$$
$$z[n] + \alpha \cdot z[n - N] = y[n]$$

So the overall difference equation is

$$z[n] + \alpha \cdot z[n - N] = x[n] + \alpha \cdot x[n - N]$$

To verify that they are the inverse of each other, we calculate their h[n]

$$h_1[n] = \delta[n] + \alpha \cdot \delta[n - N]$$
$$h_2[n] + \alpha \cdot h_2[n - N] = \delta[n]$$

Then we have

$$\begin{aligned} h_1[n] * h_2[n] &= h_2[n] * \delta[n] + \alpha \cdot h_2[n]\delta[n - N] \\ &= h_2[n] + \alpha \cdot h_2[n - N] \\ &= \delta[n] \end{aligned}$$

So we can see the two systems **are the inverse of each other**. (In addition, in Fig 2.10.b is the result of a test: Let random signal x[n] go through System 1 and 2 in order. Compare the final output z[n] with x[n] and we find that x[n] = z[n].)

(2) When z[n] = x[n], we get

$$z[n] + \alpha \cdot z[n - N] = x[n] + \alpha \cdot x[n - N]$$

So z[n] = x[n] is **a valid solution**.

Intermediate Problems

(c). The echo removal system Eq. (2.22) will have an infinite-length impulse response. Assuming that $N = 1000$, and $\alpha = 0.5$, compute the impulse response using `filter` with an input that is an impulse given by `d=[1 zeros(1,4000)]`. Store this 4001 sample approximation to the impulse response in the vector `her`.

Solution (c):

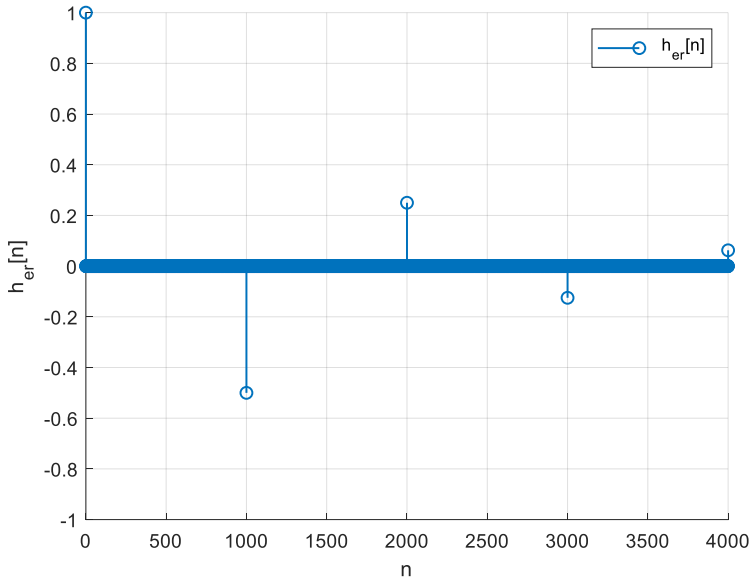


Fig 2.10.c

Analysis (c): her[h] is shown in Fig 2.10.c, by use delta[n] as the input signal into the system described by Eq. (2.22).

(d). Implement the echo removal system using `z=filter(1,a,y)`, where `a` is the appropriate coefficient vector derived from Eq. (2.22). Plot the output using `plot`. Also, listen to the output using `sound`. You should no longer hear the echo.

Solution (d):

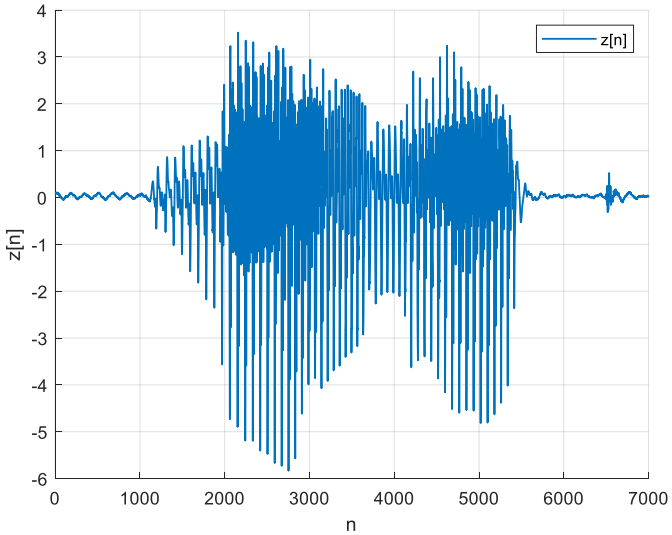


Fig 2.10.d

Analysis (d): We plot the sound wave after canceling the echo in Fig 2.10.d. And It's true that we can no longer hear the obvious echo.

(e). Calculate the overall impulse response of the cascaded echo system, Eq. (2.21), and echo removal system, Eq. (2.22), by convolving **he** with **her** and store the result in **hoa**. Plot the overall impulse response. You should notice that the result is not a unit impulse. Given that you have computed **her** to be the inverse of **he**, why is this the case?

Solution (e):

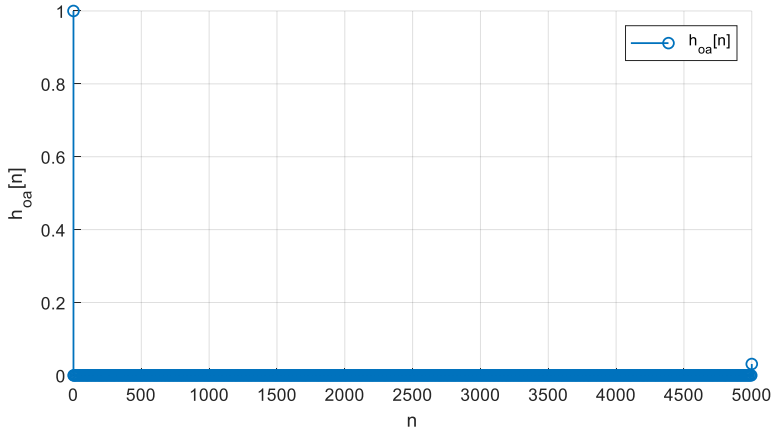


Figure 2.10.e

Analysis (e): We can see in Figure 2.10.e, hoa[5000] is not zero. That is just because the infinite impulse response *her* is **cut** to be a finite signal.

Advanced Problem

(f). Suppose that you were given $y[n]$ but did not know the value of the echo time, N, or the amplitude of the echo, α. Based on Eq. (2.21), can you determine a method of estimating these values? Hint: Consider the output **y** of the echo system to be of the form:

$$y[n] = x[n] * (\delta[n] + \alpha\delta[n - N])$$

and consider the signal,

$$R_{yy}[n] = y[n] * y[-n].$$

This is called the autocorrelation of the signal $y[n]$ and is often used in applications of echo-time estimation. Write $R_{yy}[n]$ in terms of $R_{xx}[n]$ and also plot $R_{yy}[n]$. You will have to truncate $y[n]$ before your calculations to keep $R_{yy}[n]$ within the limitations of the Student Edition of MATLAB. You will find that many of the properties of the autocorrelation will still hold when **y** is truncated. Also try experimenting with simple echo problems such as

```
>> NX=100;
>> x=randn(1,NX);
>> N=50;
>> alpha=0.9;
>> y=filter([1 zeros(1,N) alpha],1,x);
>> Ryy=conv(y,fliplr(y));
>> plot([-NX+1:NX-1],Ryy)
```

by varying `N,alpha`, and `NX`. Also, when you loaded `lineup.mat`, you loaded in two additional vectors. The vector `y2` contains the phrase "line up" with a different echo time N and different echo amplitude α. The vector `y3` contains the same phrase with two echoes, each with different times and amplitudes. Can you estimate N and α for `y2`, and N_1, α_1, N_2, and α_2 for `y3`?

Solution (f):

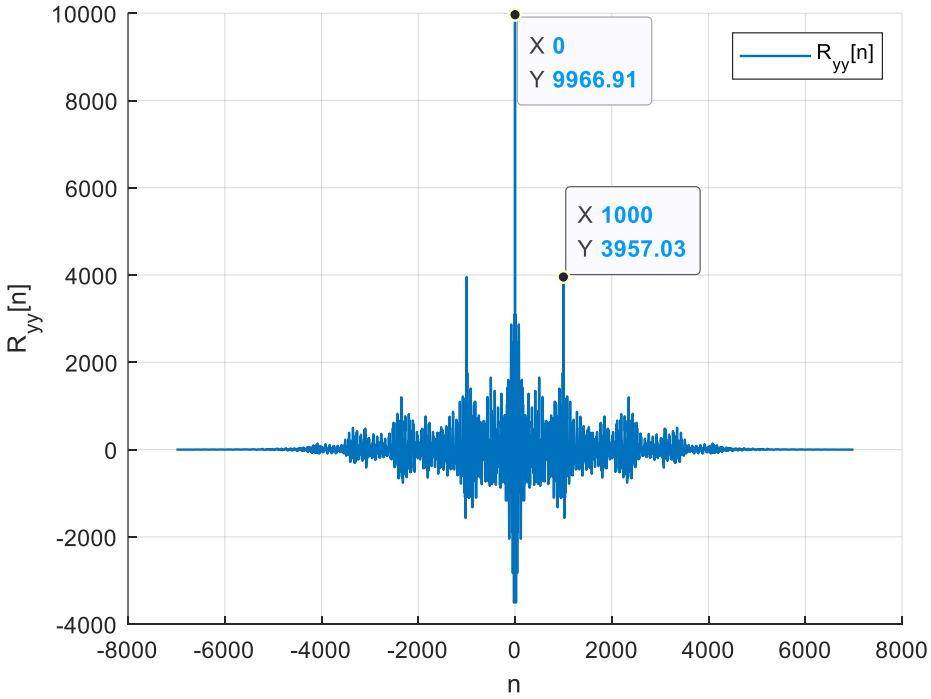


Fig 2.10.f (1), **N = 1000, α ≈ 0.494**

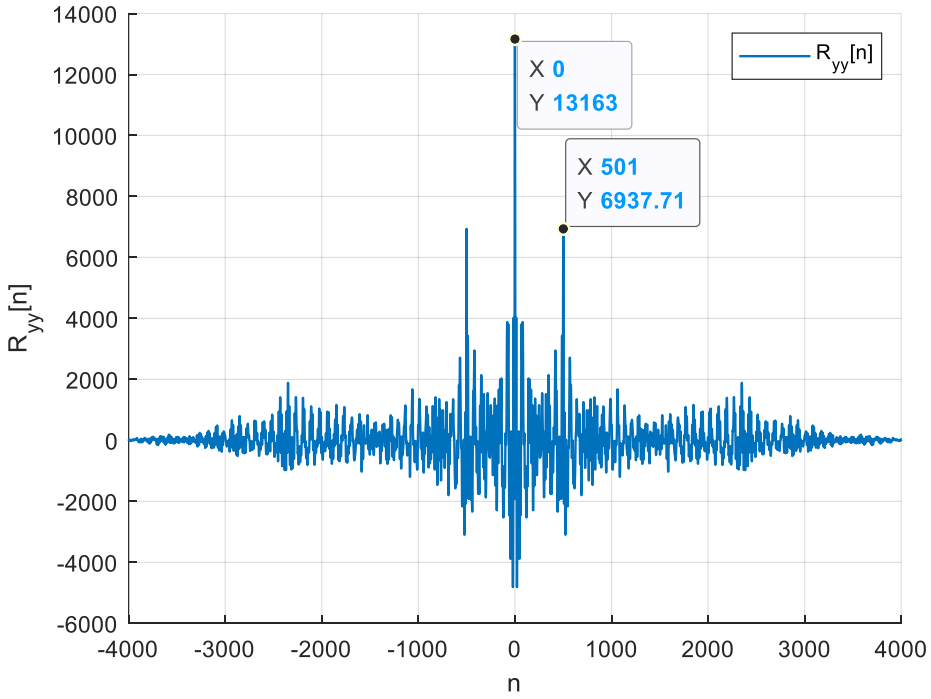


Fig 2.10.f (2), **N = 501, α ≈ 0.949 ± 0.316i**

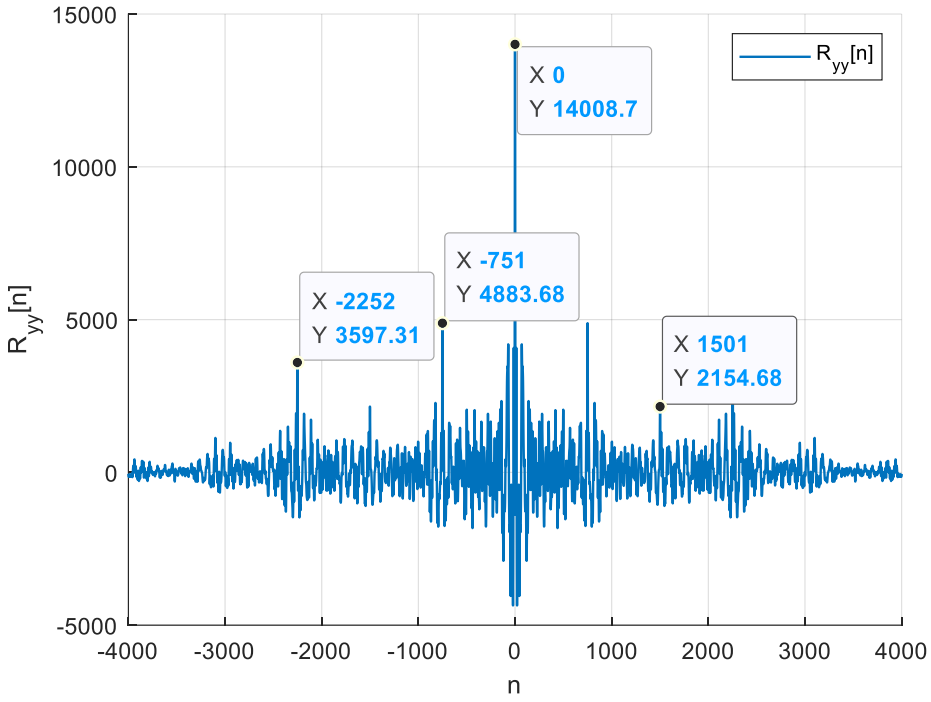


Fig 2.10.f (3) , **N1 = 751, α1 ≈ 0.599**, **N2 =2252, α2 ≈ 0.441**

Analysis (f):

(1) **Derivation**:

$$\begin{aligned} R_{yy}[n] &= y[n] * y[-n] \\ &= x[n] * (\delta[n] + \alpha\delta[n-N]) * x[-n] * (\delta[-n] + \alpha\delta[-n-N]) \\ &= R_{xx}[n] * (\delta[n] + \alpha\delta[n-N] + \alpha\delta[n+N] + \alpha^2\delta[n]) \\ &= (\alpha^2+1)R_{xx}[n] + \alpha R_{xx}[n-N] + \alpha R_{xx}[n+N] \end{aligned}$$

(2) In Fig 2.10.f (1), we pick the highest two points and use the conclusion above to solve N and α. N is the X-position of the sub peak point. (α ^ 2 + 1) / α = Y-position of the fist peak point – Y-position of the sub peak point. That is

$$\begin{cases} N = 1000 \\ \dfrac{\alpha^2+1}{\alpha} = \dfrac{9966.91}{3957.03} \end{cases}$$

So we get **N = 1000**, **α ≈ 0.494**.

(3) In Fig 2.10.f (2), do the same thing in (2) and we get **N = 501**, **α ≈ 0.949 ± 0.316i**.

(4) The new y[n]:

$$y[n] = x[n] * (\delta[n] + \alpha_1\delta[n-N_1] + \alpha_2\delta[n-N_2])$$

And then we derive the new

$$\begin{aligned} R_{yy}[n] &= \cdots \\ &= (\alpha_1^2 + \alpha_2^2 + 1)R_{xx}[n] + \alpha_1(R_{xx}[n+N_1] + R_{xx}[n-N_1]) \\ &+ \alpha_2(R_{xx}[n+N_2] + R_{xx}[n-N_2]) \\ &+ \alpha_1\alpha_2(R_{xx}[n+(N_2-N_1)] + R_{xx}[n-(N_2-N_1)]) \end{aligned}$$

Assume that $N_1 < N_2$. Now in Fig 2.10.f (3), we pick four points. Their X-positions are 0, 751, 1501, 2252. Coincidentally, we can find that 1501 + 751 = 2252. We know α < 1, so α1 * α2 < α1 or α2. So (1501, 2154.68) is the "N2 – N1" point. Then we obtain that **N1 = 751**, **N2 = 2252**.

And we have

$$\begin{cases} k\alpha_1 = 4883.68 \\ k\alpha_2 = 3597.31 \\ k\alpha_1\alpha_2 = 2154.68 \end{cases}$$

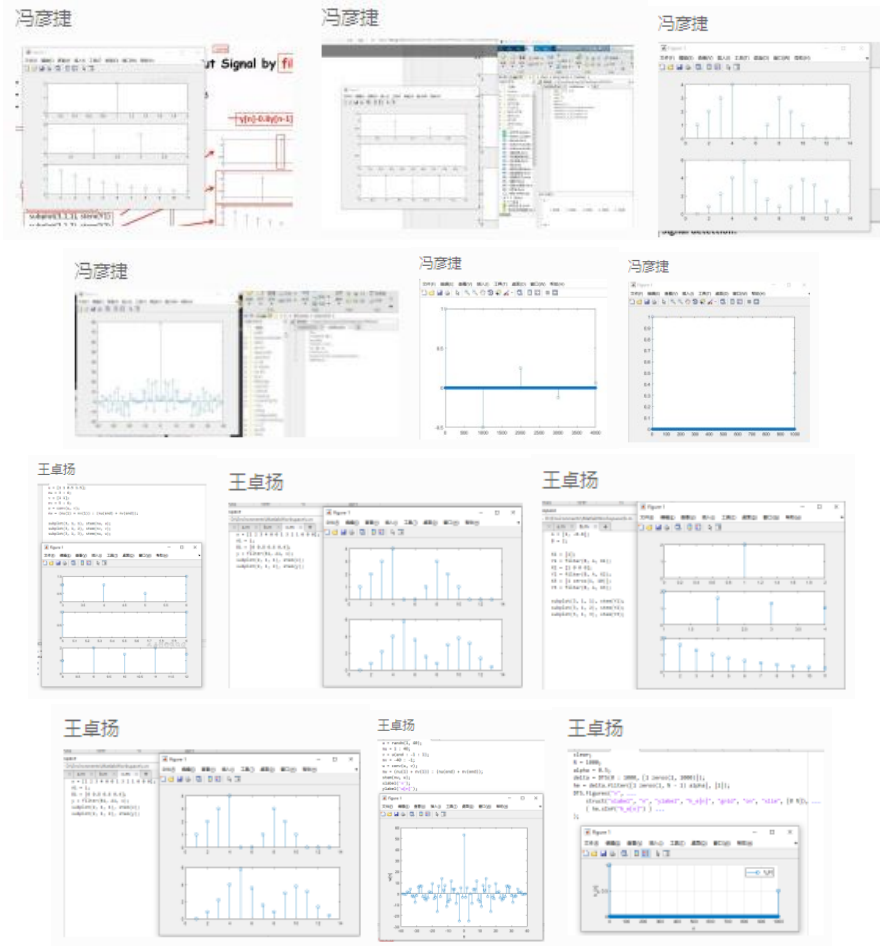
Solve them and we obtain **α1 ≈ 0.599**, **α2 ≈ 0.441**.

Note: Please indicate meaning of the symbols in all expressions. Please indicate the coordinate and unit in all figures.

Experience

1. We developed a util class *DTS* to simplify the code, which polished up the programs.
2. We learnt more about how to apply the speculative knowledge to practical experiments.
3. Next time we may need to type the formulas in better ways.
4. We understood more about how finite approximate signals influence the error between simulation and real-life conditions.
5. We program more efficient than before.
6. For the first time we solve some practical problems like echo canceling.

*** The screenshots in lab class:**



Score	99

Appendix

Util Class for DT Signal Processing – DTS.m

For the sake of simplicity, we developed a util class to help us with common tasks, including DT signal generating and storing, plotting, and doing basic operations like plus, minus, power, convolution, etc.

Here's the code in DTS.m:

```
classdef DTS
    %{
        [ 一维离散时间信号类 ]
        Latest: 20220312 by Gralerfics
        属性
            domain - 定义域，行向量
            value - 值，与 domain 等长行向量
        成员方法
            DTS(domainArg, valueArg) - 构造方法
            filter(xArgs, yArgs) - 略
            stretch(a) - x[n] --> x[a * n], a 为整且不为 0
            shift(b) - x[n] --> x[n + b], b 为整
            func(s) - 将 s.value 传入指定函数并将返回值封装为新信号
            cut(s) - 截取信号（截定义域）
            sInf([legend], [color], [style], [type]) - 生成信号绘制信息结构体
            以及一系列算符重载
                plus(+) - 叠加
                minus(-) - 削减
                uminus(-) - 取反
                times(.*) - 按元素乘
                mtimes(*) - 卷积
                rdivide(./) - 右按元素除
                power(.^) - 按元素求幂
                mpower(^) - 连续卷积
        静态方法
            Conv(a, b) - 卷积信号 a 与 b
            Filter(B, A, x) - 略
            Stems(Dims, Sigs) - 绘图
                Dims - 结构体，可含 xlim, ylim, xlabel, ylabel, title, grid 域
                Sigs - Cell 数组，元素为结构体，含 signal, legend, style, color
            Figures(type, Dims1, Sigs1, ...) - 多子图排列绘图
                type – v 纵列，h 横排
    %}
    properties
        domain
        value
    end
```

```
    methods
        function obj = DTS(domainArg, valueArg)
            obj.domain = domainArg;
            obj.value = valueArg;
        end
        function y = filter(obj, xArgs, yArgs)
            y = DTS.Filter(xArgs, yArgs, obj);
        end
        function y = stretch(obj, a)
            yDomain = obj.domain(mod(obj.domain, a) == 0) ./ a;
            yValue = obj.value(mod(obj.domain, a) == 0);
            if a < 0
                yDomain = fliplr(yDomain);
                yValue = fliplr(yValue);
            end
            y = DTS(yDomain, yValue);
        end
        function y = shift(obj, b)
            y = DTS((obj.domain(1) - b) : (obj.domain(end) - b), obj.value);
        end
        function y = func(obj, s)
            yDomain = obj.domain;
            yValue = eval(s + "(obj.value);");
            y = DTS(yDomain, yValue);
        end
        function y = cut(obj, s)
            yDomain = s;
            l = max(obj.domain(1) - s(1), 0);
            r = max(s(end) - obj.domain(end), 0);
            yValue = [zeros(1, l) obj.value(ismember(obj.domain, s)) zeros(1,
r)];
            y = DTS(yDomain, yValue);
        end
        function rst = sInf(obj, varargin)
            rst.signal = obj;
            if nargin > 1; if varargin{1} ~= ""; rst.legend = varargin{1}; end;
end
            if nargin > 2; if varargin{2} ~= ""; rst.color = varargin{2}; end;
end
            if nargin > 3; if varargin{3} ~= ""; rst.style = varargin{3}; end;
end
            if nargin > 4; if varargin{4} ~= ""; rst.type = varargin{4}; end;
end
        end
```

```
        function y = plus(a, b)
            if class(a) == "double"; a = DTS(b.domain, a * ones(1,
length(b.domain))); end
            if class(b) == "double"; b = DTS(a.domain, b * ones(1,
length(a.domain))); end
            yDomain = min(a.domain(1), b.domain(1)) : max(a.domain(end),
b.domain(end));
            yValue = zeros(1, length(yDomain));
            y = DTS(yDomain, yValue);
            s = find(y.domain == a.domain(1)); i = s : (s + length(a.value) -
1); y.value(i) = y.value(i) + a.value;
            s = find(y.domain == b.domain(1)); i = s : (s + length(b.value) -
1); y.value(i) = y.value(i) + b.value;
        end
        function y = minus(a, b)
            y = plus(a, uminus(b));
        end
        function y = uminus(a)
            yDomain = a.domain;
            yValue = -a.value;
            y = DTS(yDomain, yValue);
        end
        function y = times(a, b)
            if class(a) == "double"; a = DTS(b.domain, a * ones(1,
length(b.domain))); end
            if class(b) == "double"; b = DTS(a.domain, b * ones(1,
length(a.domain))); end
            yDomain = min(a.domain(1), b.domain(1)) : max(a.domain(end),
b.domain(end));
            yValue = zeros(1, length(yDomain));
            y = DTS(yDomain, yValue);
            s = find(y.domain == a.domain(1)); i = s : (s + length(a.value) -
1); y.value(i) = y.value(i) + a.value;
            s = find(y.domain == b.domain(1)); i = s : (s + length(b.value) -
1); y.value(i) = y.value(i) .* b.value;
        end
        function y = mtimes(a, b)
            y = DTS.Conv(a, b);
        end
        function y = rdivide(a, b)
            if class(a) == "double"; a = DTS(b.domain, a * ones(1,
length(b.domain))); end
            if class(b) == "double"; b = DTS(a.domain, b * ones(1,
length(a.domain))); end
```

```
            yDomain = min(a.domain(1), b.domain(1)) : max(a.domain(end),
b.domain(end));
            yValue = zeros(1, length(yDomain));
            y = DTS(yDomain, yValue);
            s = find(y.domain == a.domain(1)); i = s : (s + length(a.value) -
1); y.value(i) = y.value(i) + a.value;
            s = find(y.domain == b.domain(1)); i = s : (s + length(b.value) -
1); y.value(i) = y.value(i) ./ b.value;
        end
        function y = power(a, b)
            if class(a) == "double"; a = DTS(b.domain, a * ones(1,
length(b.domain))); end
            if class(b) == "double"; b = DTS(a.domain, b * ones(1,
length(a.domain))); end
            yDomain = a.domain;
            yValue = a.value .^ b.value;
            y = DTS(yDomain, yValue);
        end
        function y = mpower(a, b)
            y = DTS(a.domain, a.value);
            for i = 2 : b
                y = y * a;
            end
        end
    end
    methods(Static)
        function z = Conv(x, y)
            zDomain = (x.domain(1) + y.domain(1)) : (x.domain(end) +
y.domain(end));
            zValue = conv(x.value, y.value);
            z = DTS(zDomain, zValue);
        end
        function y = Filter(xArgs, yArgs, x)
            y = DTS(x.domain, filter(xArgs, yArgs, x.value));
        end
        function Stems(Dims, Sigs)
            hold on;
            for f = ["xlim", "ylim", "xlabel", "ylabel", "title", "grid"]
                if isfield(Dims, f)
                    eval(f + "(Dims." + f + ");");
                end
            end
            handles = [];
            legends = [];
```

```
    for i = 1 : length(Sigs)
        sig = Sigs{i};
        s = sig.signal;
        args = {s.domain, s.value};
        if isfield(sig, 'style')
            args = [args, {sig.style}];
        end
        if isfield(sig, 'color')
            args = [args, 'color', {sig.color}];
        end
        args = [args, {'lineWidth', 1}];
        h = nan;
        if isfield(sig, 'type')
            switch sig.type
                case "plot"
                    h = plot(args{:});
                otherwise
                    h = stem(args{:});
            end
        else
            h = stem(args{:});
        end
        if isfield(sig, 'legend')
            handles = [handles, h];
            legends = [legends, sig.legend];
        end
    end
    if ~isempty(legends)
        legend(handles, legends');
    end
end
function Figures(varargin)
    n = (nargin - 1) / 2;
    switch varargin{1}
        case "v"
            p = 100 * n + 10;
        case "h"
            p = 100 + n * 10;
    end
    for i = 1 : n
        subplot(p + i);
        DTS.Stems(varargin{i * 2}, varargin{i * 2 + 1});
    end
end
```

```
    end
end
```

Programs for 2.4

(a)

```
clear;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
h1 = DTS(0 : 4, [1 -1 3 1 0]);
h2 = DTS(0 : 4, [0 2 5 4 -1]);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "x_1[n]", "grid", "on"), ...
    { x1.sInf("x_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "h_1[n]", "grid", "on"), ...
    { h1.sInf("h_1[n]", "#2E9F79") }, ...
    struct("xlabel", "n", "ylabel", "h_2[n]", "grid", "on"), ...
    { h2.sInf("h_2[n]", "#5574C6") } ...
);
```

(b)

```
clear;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
h1 = DTS(0 : 4, [1 -1 3 1 0]);
y1 = x1 * h1;
y2 = h1 * x1;
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "x_1[n]", "grid", "on"), ...
    { x1.sInf("x_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "h_1[n]", "grid", "on"), ...
    { h1.sInf("h_1[n]", "#2E9F79") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { ...
        y1.sInf("x_1[n] * h_1[n]", "", "b"), ...
        y2.sInf("h_1[n] * x_1[n]", "", "g--") ...
    } ...
);
```

(c)

```
clear;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
h1 = DTS(0 : 4, [1 -1 3 1 0]);
h2 = DTS(0 : 4, [0 2 5 4 -1]);
y1 = x1 * (h1 + h2);
y2 = x1 * h1 + x1 * h2;
DTS.Figures("v", ...
```

```
    struct("xlabel", "n", "ylabel", "x_1[n]", "grid", "on"), ...
    { x1.sInf("x_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "h_1[n]", "grid", "on"), ...
    { h1.sInf("h_1[n]", "#2E9F79") }, ...
    struct("xlabel", "n", "ylabel", "h_2[n]", "grid", "on"), ...
    { h2.sInf("h_2[n]", "#2E9F79") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { ...
        y1.sInf("x_1[n] * (h_1[n] + h_2[n])", "", "b"), ...
        y2.sInf("x_1[n] * h_1[n] + x_1[n] * h_2[n]", "", "g--") ...
    } ...
);
```

(d)

```
clear;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
h1 = DTS(0 : 4, [1 -1 3 1 0]);
h2 = DTS(0 : 4, [0 2 5 4 -1]);
w = x1 * h1;
yd1 = w * h2;
hs = h1 * h2;
yd2 = x1 * hs;
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "w[n]", "grid", "on"), ...
    { w.sInf("w[n] = x_1[n] * h_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "y_{d1}[n]", "grid", "on"), ...
    { yd1.sInf("y_{d1}[n] = w[n] * h_2[n]", "#2E9F79") }, ...
    struct("xlabel", "n", "ylabel", "h_{series}[n]", "grid", "on"), ...
    { hs.sInf("h_{series}[n] = h_1[n] * h_2[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "y_{d2}[n]", "grid", "on"), ...
    { yd2.sInf("y_{d2}[n] = x_1[n] * h_{series}[n]", "#2E9F79") } ...
);
```

(e)

```
clear;
n0 = 2;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
he1 = DTS(0 : 4, [1 -1 3 1 0]);
he2 = he1.shift(-n0);
ye1 = x1 * he1;
ye2 = x1 * he2;
ye1s = ye1.shift(-n0);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
```

```
    { ye1.sInf("y_{e1}[n] = x_1[n] * h_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on", "xlim", [0 14]), ...
    { ye2.sInf("y_{e2}[n] = x_1[n] * h_1[n - n_0]", "#2E9F79") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on", "xlim", [0 14]), ...
    { ye1s.sInf("y_{e1}[n - n_0]", "#5574C6") } ...
);
```

(f)

```
clear;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
hf2 = DTS(0 : 4, [1 -1 3 1 0]);
np1 = DTS(0 : 9, 1 : 10); % (n + 1)
w = np1 .* x1; % Step 1
yf1 = w * hf2; % Step 2
delta = DTS(0 : 4, [1 0 0 0 0]); % Step 3
hf1 = np1.cut(0 : 4) .* delta;
hseries = hf1 * hf2; % Step 4
yf2 = x1 * hseries; % Step 5
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { yf1.sInf("y_{f1}[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { yf2.sInf("y_{f2}[n]", "#2E9F79") } ...
);
```

(g)

```
clear;
x1 = DTS(0 : 9, [ones(1, 5) zeros(1, 5)]);
hg2 = DTS(0 : 4, [0 2 5 4 -1]);
xg = DTS(0 : 4, [2 0 0 0 0]); % Step 1
yga = xg .^ 2;
ygb = xg * hg2; % Step 2
yg1 = yga + ygb; % Step 3
delta = DTS(0 : 4, [1 0 0 0 0]); % Step 4
hg1 = delta .^ 2;
hparallel = hg1 + hg2; % Step 5
yg2 = xg * hparallel; % Step 6
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { yg1.sInf("y_{g1}[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { yg2.sInf("y_{g2}[n]", "#2E9F79") } ...
);
```

Programs for 2.5

(a)

```
clear;
x1 = DTS(-1 : 5, [0 1 0 0 0 0 0]);
x2 = x1.shift(-1);
x3 = x1 + 2 .* x2;
w1 = x1.filter([1 -1 -1], [1]).cut(0 : 5);
w2 = x2.filter([1 -1 -1], [1]).cut(0 : 5);
w3 = x3.filter([1 -1 -1], [1]).cut(0 : 5);
w4 = w1 + 2 .* w2;
y1 = x1.func('cos').cut(0 : 5);
y2 = x2.func('cos').cut(0 : 5);
y3 = x3.func('cos').cut(0 : 5);
y4 = y1 + 2 .* y2;
n = DTS(0 : 5, 0 : 5);
z1 = n .* x1; z1 = z1.cut(0 : 5);
z2 = n .* x2; z2 = z2.cut(0 : 5);
z3 = n .* x3; z3 = z3.cut(0 : 5);
z4 = z1 + 2 .* z2;
figure(1);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "w[n]", "grid", "on"), ...
    { w1.sInf("w_1[n]", "#331199") }, ...
    struct("xlabel", "n", "ylabel", "w[n]", "grid", "on"), ...
    { w2.sInf("w_2[n]", "#993366") }, ...
    struct("xlabel", "n", "ylabel", "w[n]", "grid", "on"), ...
    { w3.sInf("w_3[n]", "#CC4488") }, ...
    struct("xlabel", "n", "ylabel", "w[n]", "grid", "on"), ...
    { w4.sInf("w_1[n] + 2w_2[n]", "#FFAA55") } ...
);
figure(2);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { y1.sInf("y_1[n]", "#331199") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { y2.sInf("y_2[n]", "#993366") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { y3.sInf("y_3[n]", "#CC4488") }, ...
    struct("xlabel", "n", "ylabel", "y[n]", "grid", "on"), ...
    { y4.sInf("y_1[n] + 2y_2[n]", "#FFAA55") } ...
);
figure(3);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on"), ...
```

```
    { z1.sInf("z_1[n]", "#331199") }, ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on"), ...
    { z2.sInf("z_2[n]", "#993366") }, ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on"), ...
    { z3.sInf("z_3[n]", "#CC4488") }, ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on"), ...
    { z4.sInf("z_1[n] + 2z_2[n]", "#FFAA55") } ...
);
```

(d)

```
clear;
delta = DTS(0 : 19, [1 zeros(1, 19)]);
h1 = delta.filter([1], [1 -0.6]);
n = DTS(0 : 19, 0 : 19);
h2 = 0.6 .^ (n .* (n + 1) ./ 2);
figure(1);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "h_1[n]", "grid", "on", "xlim", [0 19]), ...
    { h1.sInf("h_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "h_2[n]", "grid", "on", "xlim", [0 19]), ...
    { h2.sInf("h_2[n]", "#2E9F79") } ...
);
```

(e)

```
clear;
step = DTS(0 : 19, ones(1, 20));
s1 = step.filter([1], [1 -0.6]);
s2 = step;
for i = 2 : 20
    s2.value(i) = 0.6 ^ (i - 1) * s2.value(i - 1) + s2.value(i);
end
figure(1);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "s_1[n]", "grid", "on", "xlim", [0 19]), ...
    { s1.sInf("s_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "s_2[n]", "grid", "on", "xlim", [0 19]), ...
    { s2.sInf("s_2[n]", "#2E9F79") } ...
);
```

(f)

```
clear;
delta = DTS(0 : 19, [1 zeros(1, 19)]);
step = DTS(0 : 19, ones(1, 20));
h1 = delta.filter([1], [1 -0.6]);
```

```
n = DTS(0 : 19, 0 : 19);
h2 = 0.6 .^ (n .* (n + 1) ./ 2);
z1 = h1 * step; z1 = z1.cut(0 : 19);
z2 = h2 * step; z2 = z2.cut(0 : 19);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "z_1[n]", "grid", "on", "xlim", [0 19]), ...
    { z1.sInf("z_1[n]", "#01AFEE") }, ...
    struct("xlabel", "n", "ylabel", "z_2[n]", "grid", "on", "xlim", [0 19]), ...
    { z2.sInf("z_2[n]", "#2E9F79") } ...
);
```

(g)

```
clear;
delta = DTS(0 : 19, [1 zeros(1, 19)]);
step = DTS(0 : 19, ones(1, 20));
s1 = step.filter([1], [1 -0.6]);
s2 = step;
for i = 2 : 20
    s2.value(i) = 0.6 ^ (i - 1) * s2.value(i - 1) + s2.value(i);
end
h1 = delta.filter([1], [1 -0.6]);
n = DTS(0 : 19, 0 : 19);
h2 = 0.6 .^ (n .* (n + 1) ./ 2);
z1 = h1 * step; z1 = z1.cut(0 : 19);
z2 = h2 * step; z2 = z2.cut(0 : 19);
figure(1);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "s_1[n] & z_1[n]", "grid", "on", "xlim",
[0 19]), ...
    { s1.sInf("s_1[n]"), z1.sInf("z_1[n]") } ...
);
figure(2);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "s[n]", "grid", "on", "xlim", [0 19]), ...
    { s2.sInf("s_2[n]") }, ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on", "xlim", [0 19]), ...
    { z2.sInf("z_2[n]") } ...
);
```

Programs for 2.10

(a)

```
clear;
N = 1000;
alpha = 0.5;
```

```
delta = DTS(0 : 1000, [1 zeros(1, 1000)]);
he = delta.filter([1 zeros(1, N - 1) alpha], [1]);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "h_e[n]", "grid", "on", "xlim", [0 N]), ...
    { he.sInf("h_e[n]") } ...
);
```

(b)

```
clear;
N = 1000;
alpha = 0.5;
x = DTS(0 : 1000, [1 rand(1, 1000) .* rand(1, 1000)]);
y = x.filter([1 zeros(1, N - 1) alpha], [1]);
z = y.filter([1], [1 zeros(1, N - 1) alpha]);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "x[n]", "grid", "on", "xlim", [0 N]), ...
    { x.sInf("x[n]", "#01AFEE", ".") }, ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on", "xlim", [0 N]), ...
    { z.sInf("z[n]", "#2E9F79", ".") } ...
);
```

(c)

```
clear;
N = 1000;
alpha = 0.5;
d = DTS(0 : 4000, [1 zeros(1, 4000)]);
her = d.filter([1], [1 zeros(1, N - 1) alpha]);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "h_{er}[n]", "grid", "on", "ylim", [-1
1]), ...
    { her.sInf("h_{er}[n]") } ...
);
```

(d)

```
clear;
N = 1000;
alpha = 0.5;
load("lineup.mat");
s = DTS(0 : (length(y) - 1), y');
z = s.filter([1], [1 zeros(1, N - 1) alpha]);
sound(z.value, 8192);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "z[n]", "grid", "on"), ...
    { z.sInf("z[n]", "", "", "plot") } ...
```

```
);
```

(e)

```
clear;
N = 1000;
alpha = 0.5;
delta = DTS(0 : 1000, [1 zeros(1, 1000)]);
he = delta.filter([1 zeros(1, N - 1) alpha], [1]);
d = DTS(0 : 4000, [1 zeros(1, 4000)]);
her = d.filter([1], [1 zeros(1, N - 1) alpha]);
hoa = he * her;
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "h_{oa}[n]", "grid", "on"), ...
    { hoa.sInf("h_{oa}[n]") } ...
);
```

(f)

```
clear;
load("lineup.mat");
Y = DTS(0 : (length(y) - 1), y');
Ryy = Y * Y.stretch(-1);
figure(1);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "R_{yy}[n]", "grid", "on"), ...
    { Ryy.sInf("R_{yy}[n]", "", "", "plot") } ...
);

Y = DTS(0 : (length(y2) - 1), y2');
Ryy = Y * Y.stretch(-1);
figure(2);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "R_{yy}[n]", "grid", "on", "xlim", [-4000
4000]), ...
    { Ryy.sInf("R_{yy}[n]", "", "", "plot") } ...
);

Y = DTS(0 : (length(y3) - 1), y3');
Ryy = Y * Y.stretch(-1);
figure(3);
DTS.Figures("v", ...
    struct("xlabel", "n", "ylabel", "R_{yy}[n]", "grid", "on", "xlim", [-4000
4000]), ...
    { Ryy.sInf("R_{yy}[n]", "", "", "plot") } ...
);
```