NAME - SANTOSH (CB.EN.U4CCE20053)

DEPARTMENT - COMPUTER AND COMMUNICATION ENGINEERING (CCE) A

LAB TITLE AND CODE : SIGNAL PROCESSING LAB 19CCE281

EXPERIMENT NUMBER : 4

DATE : 05/10/2021

PROPERTIES OF SYSTEM

* **AIM :**

To determine whether a given input sequence is a Linear Time-Invariant (LTI) System or not.

* **SOFTWARE REQUIRED :**

Spyder IDE (Anaconda 3) - Python 3.9.1 (64-bit)

* **THEORY : (LINEARITY)**

The principle of linearity is equivalent to the principle of superposition, i.e, a system can be said to be linear if, for any two input signals, their linear combination yields as output the same linear combination of the corresponding output signals.

→ **Definition -**

Given any two inputs, acted on by the system as follows:

$x_1(t) \rightarrow S \rightarrow y_1(t)$; $x_2 \rightarrow S \rightarrow y_2(t)$

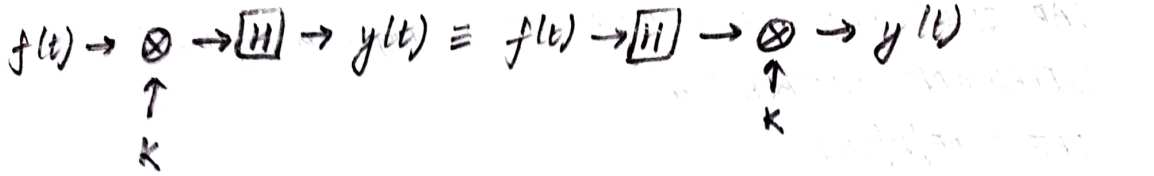
The system is said to be linear if, for any two constants α and β, and the signal : $x_3(t) = \alpha x_1(t) + \beta x_2(t)$.

The system acts on it as follows:

$x_1(t) \rightarrow S \rightarrow y(t)$, where $y(t) = \alpha y_1(t) + \beta y_2(t)$

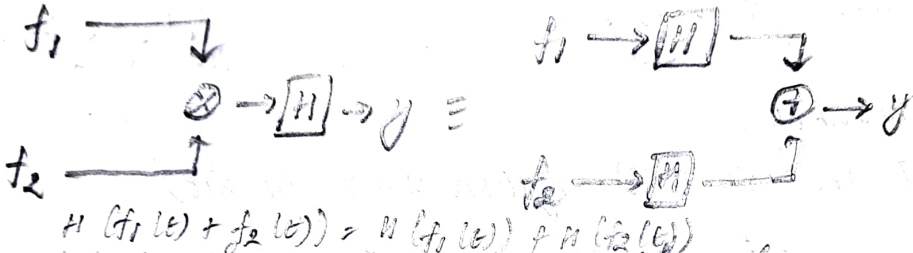
It is not necessary for the input and output signals to have the same independent variable for linearity to make sense. The definition for systems with input and/or output signal being discrete-time is similar.

PTO



$H(Kf(t)) = KH(f(t))$ → 1st step where K is the scaling factor

A block diagram demonstrating the scaling property of linearity.



$H(f_1(t) + f_2(t)) = H(f_1(t)) + H(f_2(t))$

A block diagram demonstrating the superposition property of linearity.

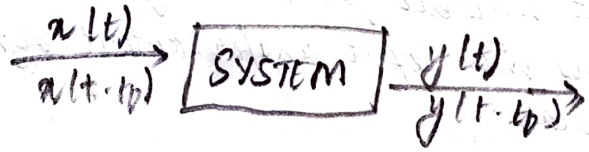
$$H(k_1 f_1(t) + k_2 f_2(t)) = k_1 H(f_1(t)) + k_2 H(f_2(t))$$

→ Example -

A capacitor, an inductor, a resistor or any combination of these are all linear systems. If we consider the voltage applied across them as an input signal, and the current through them as an output signal. This is because these simple passive circuit components follow the principle of superposition within the ranges of operation.

* THEORY: (TIME INVARIANCE)

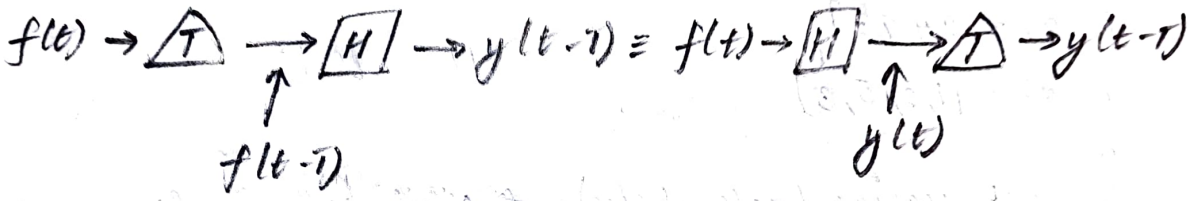
In a time-invariant system, there are no changes in system structure as a function of a time t. A system is time-invariant if -



PTO

From the diagram, we can observe that for any time to and any input x(t) that produces response y(t), the response to the time-shifted input x(t-to) is equal to the time-shifted output y(t-to) of y(t).

Therefore, in a time-invariant system, the response to a left or right shift of the input x(t) is equal to a corresponding left or right shift in the response y(t).



This block diagram shows the condition for time invariance. The output is the same whether the delay is put on the input or the output.

→ Example -
An electric circuit with fixed values of Resistance R, Inductance L, and capacitance C can be considered as time invariant.

+ GRAPH PLOTTING ALGORITHM :
The followed steps were followed -

① Define the x-axis and corresponding y-axis as lists.
② Plot them on canvas using .plot() function.
③ Give a name to x-axis and y-axis using .xlabel() and .ylabel() functions.
④ Give a title to your plot using the .title() function.
⑤ Finally, to view your plot, we use the .show() function.

PTO

PROGRAM WITH COMMENTS:

```
# import library source files
import numpy as np
import matplotlib.pyplot as plt

n = np.arange(0,4) # Get evenly spaced values within the interval [0,4)
#Given Input Sequences:
x1 = [2,4,1,3]
x2 = [1,2,5,3]

def scaling(scale, choice): # check Homogeneity condition
    before_scaling = []; after_scaling = []

if choice == str(1): # Calculate for x1 = [2,4,1,3]

    for sample in range(len(n)):
        temp = sample * x1[sample]
        before_scaling.append(int(scale) * temp) # Scaling the output

for sample in range(len(n)):
    temp = int(scale) * x1[sample]
    after_scaling.append(sample * temp) # Sending output into system

else: # Calculate for x2 = [1,2,5,3]

for sample in range(len(n)):
    temp = sample * x2[sample]
    before_scaling.append(int(scale) * temp) # Scaling the output
```

```
30      for sample in range (len(n)):
31          temp = int (scale) * x2[sample]
32          after_scaling. append (sample * temp) # Sending
33  output into system
34    plt.subplot (3,1,1) # subplot (rows, columns, index
    describes the figure layout
35    plt. xlabel ('n')
36    plt. ylabel ('y' + str (choice) + '[n]')
37.   plt. title ('Before scaling x' + str (choice) + '{}'. format
    (before_scaling) # Formats the specified value(s) and insert
    the string's placeholder.
38    plt. stem (n, before_scaling) # stem plot plots vertical
    lines at each x position covered under the graph from the
    baseline to y, and places a marker there.
39
40    plt. subplot (3, 1, 3) # subplot (rows, columns, index) describes
    the figure layout
41    plt. xlabel ('n')
42    plt. ylabel ('y' + str (choice) + '[n]')
43    plt. title ('After scaling x' + str (choice) + '{}'. format
    (after_scaling)) # Formats the specified value(s) and insert
    them inside the string's placeholder.
44    plt. stem (n, after_scaling) # stem plot plots vertical
    lines at each x position covered under the graph from the
    baseline to y, and places a marker there.
45
46    plt. show() # To view both the sub-plots
47
48    if (before_scaling == after. scaling): # Condition to check if
    both the systems are equal or not after scaling
49        print (" The given system satisfies Homogeneity condition.")
50        return (1)
```

P.T.O

```
    else:
        print (" The given system does not satisfy Homogeneity
Homogeneity condition.")
        return (0)

def superposition (): # check additivity condition
    before_adding = []; after_adding = []; y1 = []; y2 = []; x = []

    for sample in range (len(n)):
        y1.append (sample * x1[sample])
        y2.append (sample * x2[sample])
        before_adding.append (y1[sample] + y2[sample])
# Processing System Individually

    plt.subplot (3,1,1) # subplot (rows, columns, index) describes
the figure layout
    plt.xlabel ('n')
    plt.ylabel ('y[n]')
    plt.title ('H (x1 (t) + x2(t))')
    plt.stem (n, before_adding)

    for sample in range (len(n)):
        x.append (x1[sample] + x2[sample])
        after_adding.append (sample * x[sample])
# Processing system Together

    plt.subplot (3,1,3) # subplot (rows, columns, index)
describes the figure layout
    plt.xlabel ('n')
    plt.ylabel ('y[n]')
    plt.title ('H (x1 (t)) + H (x2(t))')
    plt.stem (n, after_adding) # stem plot plots vertical
lines at each x position covered under the graph from the
baseline to y, and places a marker there.
```

```
78
79  plt.show() # To view both the sub-plots
80
81    print ('\n 2. Superposition Condition :- ')
82    if (before_adding == after_adding): # Condition to check if both the systems are equal or not after adding
83        print (" The given system satisfies superposition condition.")
84        return (1)
85    else:
86        print (" The given system does not satisfy superposition condition.")
87        return (0)
88
89  def time (shift, choice): # Check Time Variance
90      Input_delay = [] ; output_delay = []
91
92      if choice == str (1): # Calculate for x1 = [2, 4, 7, 3]
93
94          # Output Delay
95          for sample in range (len (x1)):
96              output_delay.append (sample * x1[-sample-2]) # sending output into system
97
98          for sample in range (int (shift)):
99              output_delay.append (0) # Adds extra sample at the end for shifting
100
101         for sample in range (int (shift)):
102             for sample in range (len (output_delay)-1, 0, -1):
103                 output_delay [sample] = output_delay [sample-1]
    # shift the sample
104
```

P.T.O.

```
105     for sample in range (int (shift)):
106         output_delay [sample] = 0 # Assign zero to the empty
elements for the shifted bit
107
108     # Input Delay
109     for sample in range (int (shift)):
110         x1. append (0) # Adds extra sample at the end for
shifting
111
112     for sample in range (int (shift)) :
113         for sample in range (len (x1) - 1, 0, -1) :
114             x1 [sample] = x1 [sample-1] # shift the sample
115
116     for sample in range (int (shift)) :
117         x1 [sample] = 0 # Assign zero to the empty elements
for the shifted bit
118
119     for sample in range (len (x1)):
120         input_delay. append (sample * x1 [sample]) # sending
input into system
121
122 else : # Calculate for x2=[1, 2, 5, 3]
123
124     # output Delay
125     for sample in range (len (x2)):
126         output_delay. append (sample * x2 [sample]) # sending
output into system
127
128     for sample in range (int (shift)):
129         output_delay. append (0) # Adds extra sample at
the end for shifting
130
131
```

P.T.O

```
131         for sample in range (int (shift)):
132             for sample in range (len (output_delay)-1, 0, -1):
133                 output_delay [sample] = output_delay [sample -1]
    # shift the output
134
135     for sample in range (int (shift)):
136         output_delay [sample] = 0 # Assign zero to the empty
    elements for the shifted bit
137
138     # Input Delay
139     for sample in range (int (shift)):
140         x2. append (0) # Adds extra sample at the end for shifting
141
142     for sample in range (int (shift)):
143         for sample in range (len (x2) -1, 0, -1):
144             x2 [sample] = x2 [sample -1] # shift the sample
145
146     for sample in range (int (shift)):
147         x2 [sample] = 0 # Assign zero to the empty elements for
    the shifted bit
148
149     for sample in range (len (x2)):
150         input_delay. append (sample + x2 [sample]) # sending
    input into system
151
152  plt. subplot (3,1,1) # subplot (rows, columns, index) describes
    the figure layout
153  plt. xlabel ('t')
154  plt. ylabel ('y(t)')
155  plt. title ('shifting input by ' + str (shift) + ' for n' + str (choice)
    + '{}'. format (input_delay)) # Formats the specified value(s)
    and insert them inside the string's placeholder.
```

PTO

```
156     plt.stem(np.arange(len(input_delay)), input_delay) # stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
157
158     plt.subplot(3,1,3) # subplot(rows, columns, index) describes the figure layout
159     plt.xlabel('t')
160     plt.ylabel('y(t)')
161     plt.title('shifting output by ' + str(shift) + ' for x' + str(choice) + '{3}'.format(output_delay)) # Formats the specified value(s) and insert them inside the string's placeholder.
162     plt.stem(np.arange(len(output_delay)), output_delay) # stem plot plots vertical lines at each x position covered under the graph from the baseline to y, and places a marker there.
163
164     plt.show() # To view both the sub-plots
165
166     if(input_delay == output_delay): # Condition to check if both the system are equal or not after shifting
167         print(" Since both the outputs are same, the given system is time invariant.")
168         second_condition = 1
169     else:
170         print(" Since both the outputs are not same, the given system is time variant.")
171         second_condition = 0
172
173     return(second_condition)
174
175 # Main
176 while True: # This simulates a Do Loop
```

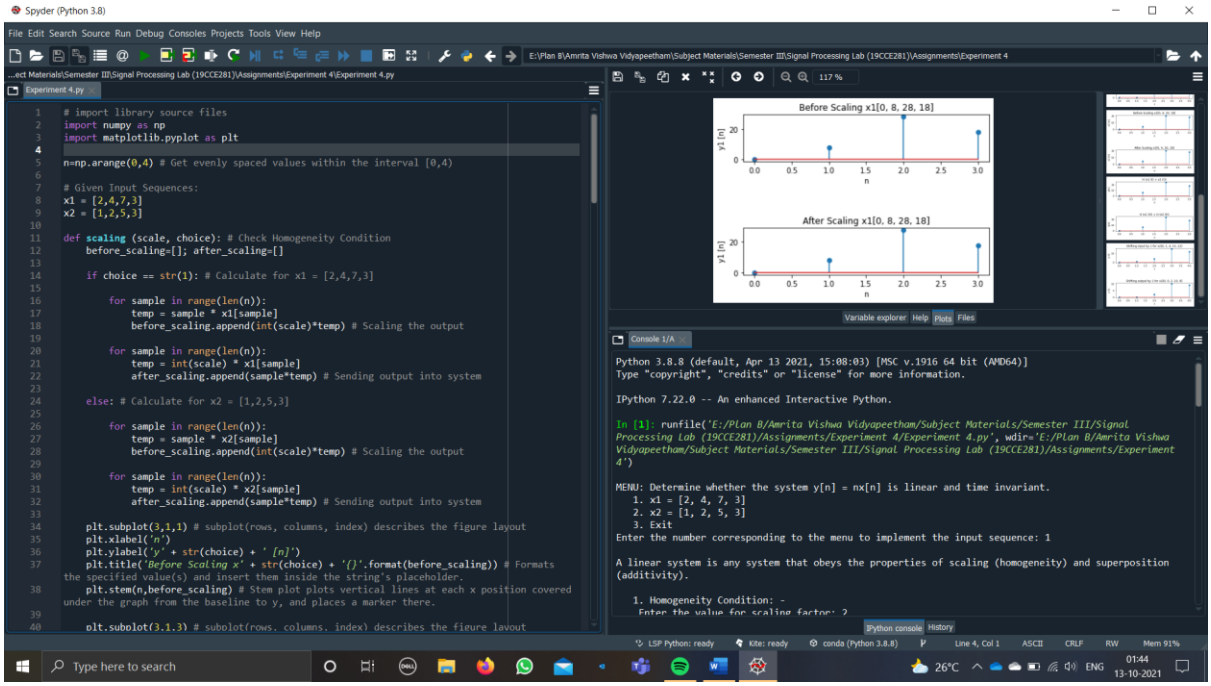
```
choice
177  ~~choice~~ = input(
178  "MENU: Determine whether the system y[n] = nx[n] is linear and time invariant.\n 1. x1 = [3,4,1,3]\n 2. x2 = [1,2,5,3]\n 3. Exit\n Enter the number corresponding to the menu to implement the input sequence: ") # Menu Driven Implementation
179
180  if choice == str(1) or choice == str(2):
181
182      scale = input("A linear system is any system that obeys the properties of scaling (homogeneity) and superposition (additity).\n \n 1. Homogeneity Condition: -\n Enter the value for scaling factor: ")
183
184      if (scaling (scale, choice) == superposition()): # Condition to check if both the systems or linear or not
185          print("\n Since both the properties are satisfied; the given system is linear.")
186          first_condition = 1
187      else:
188          print("\n Since both the properties are not satisfied; the given system is not linear.")
189          first_condition = 0
190
191      shift = input("A system is time-invariant if a time shift (i.e., advance or delay in the input always results only in an identical time shift in the output.\n Enter the value for shifting factor: ")
192
193      if (first_condition + time (shift, choice)==2): # Condition to check for LTI system or not
194          print("\n The given system is a linear Time Invariant (LTI) system.")
195      else:
```

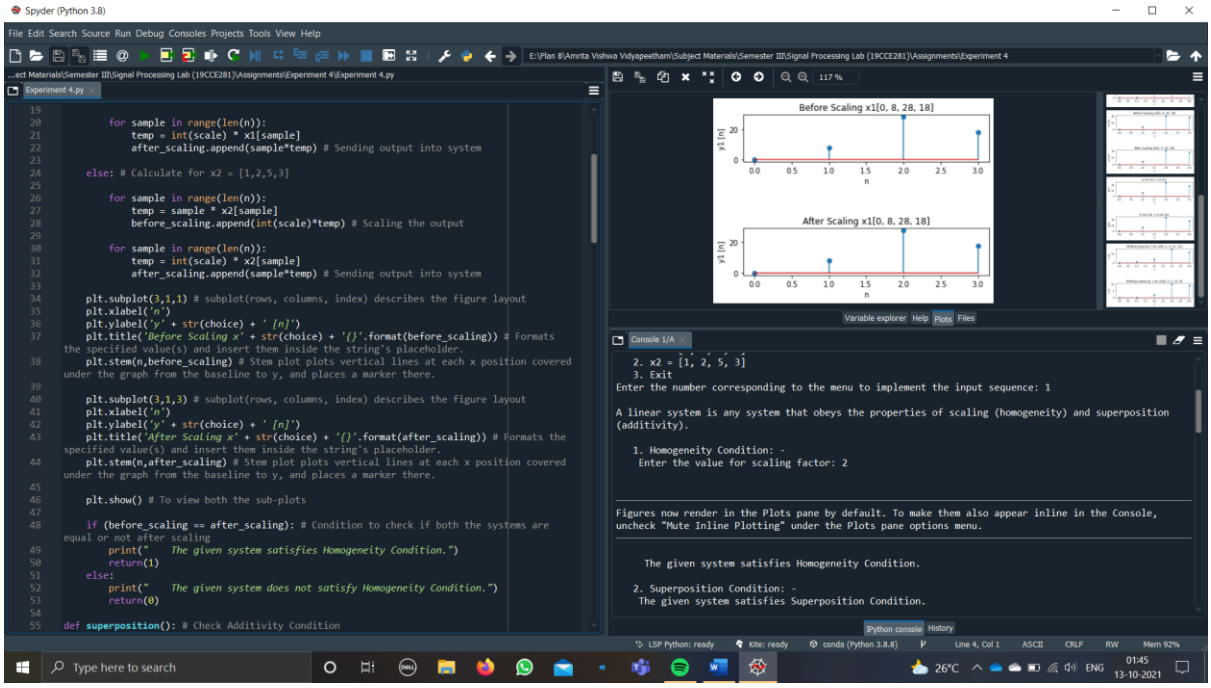
```
196         print("\n The given system is not a Linear Time Invariant (LTI) system.")
197
198     break
199
200 elif choice == str(3):
201         break : # Exit Loop
202
203 else:
204     print("Error: Invalid Input! Please try again.\n")
```

+ INFERENCE:

Examine the given time sequence as Linear Time Invariant (LTI) system or not using Python code and results verified.

Check Homogeneity Condition





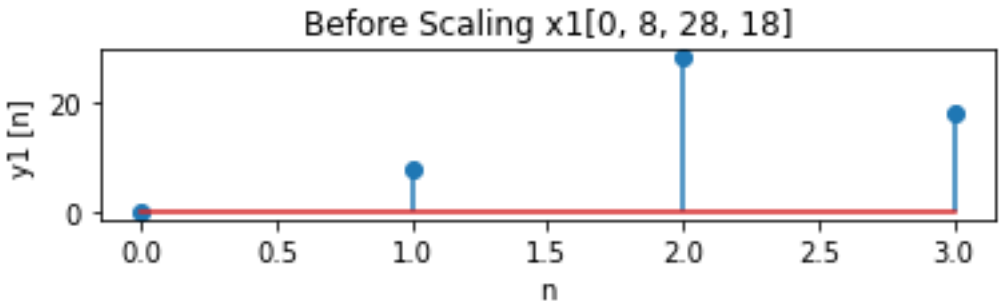
Step 1: Scale the output.

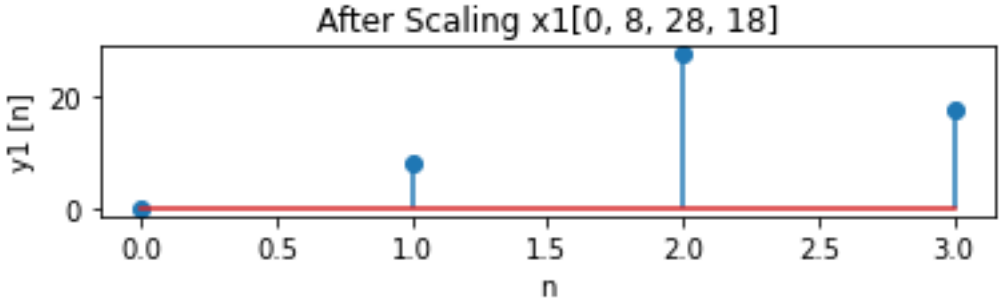
Step 2: Send output into the system.

Step 3: Plot graphs before and after scaling.

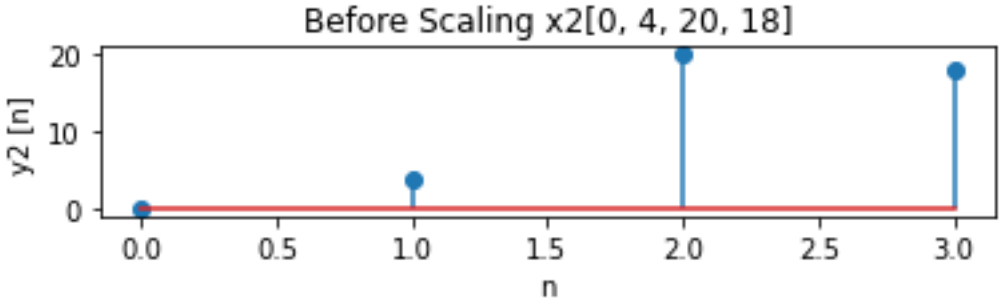
Step 4: Check if both the systems are equal or not after scaling.

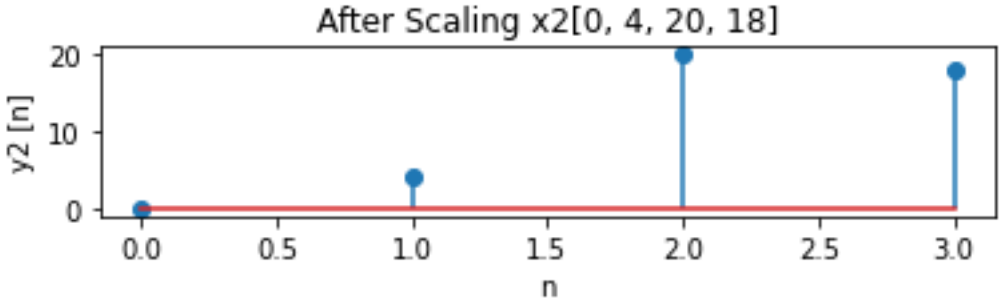
x1 = [2,4,7,3]





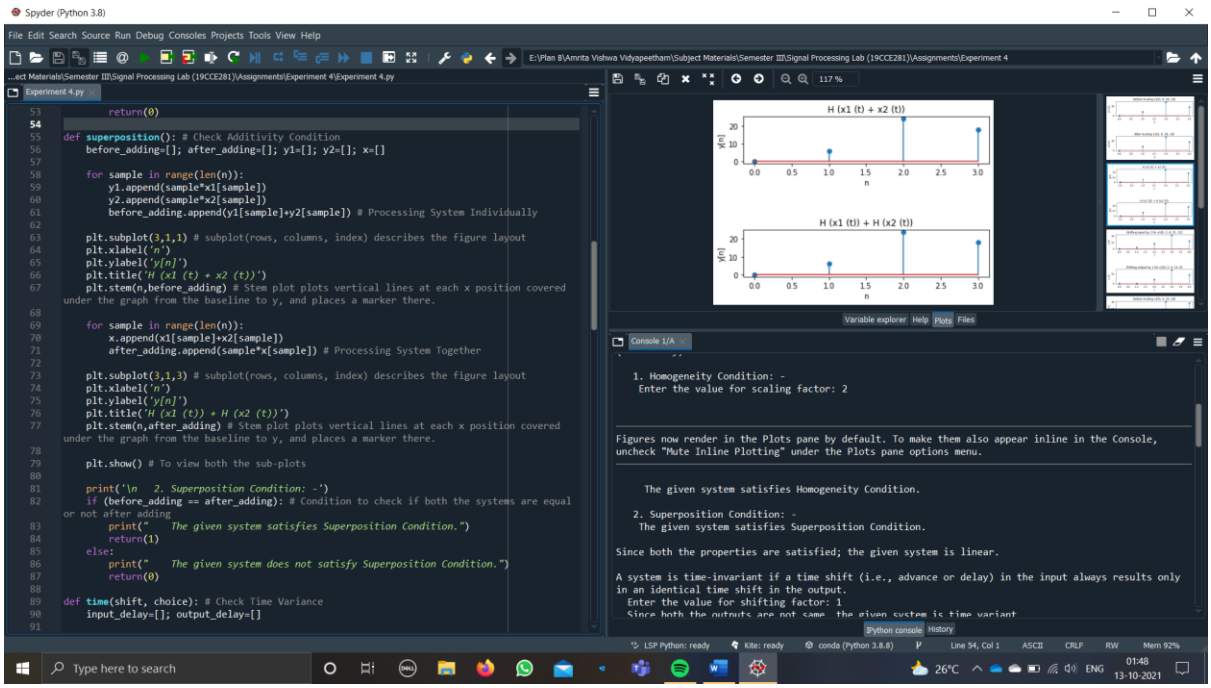
x2 = [1,2,5,3]

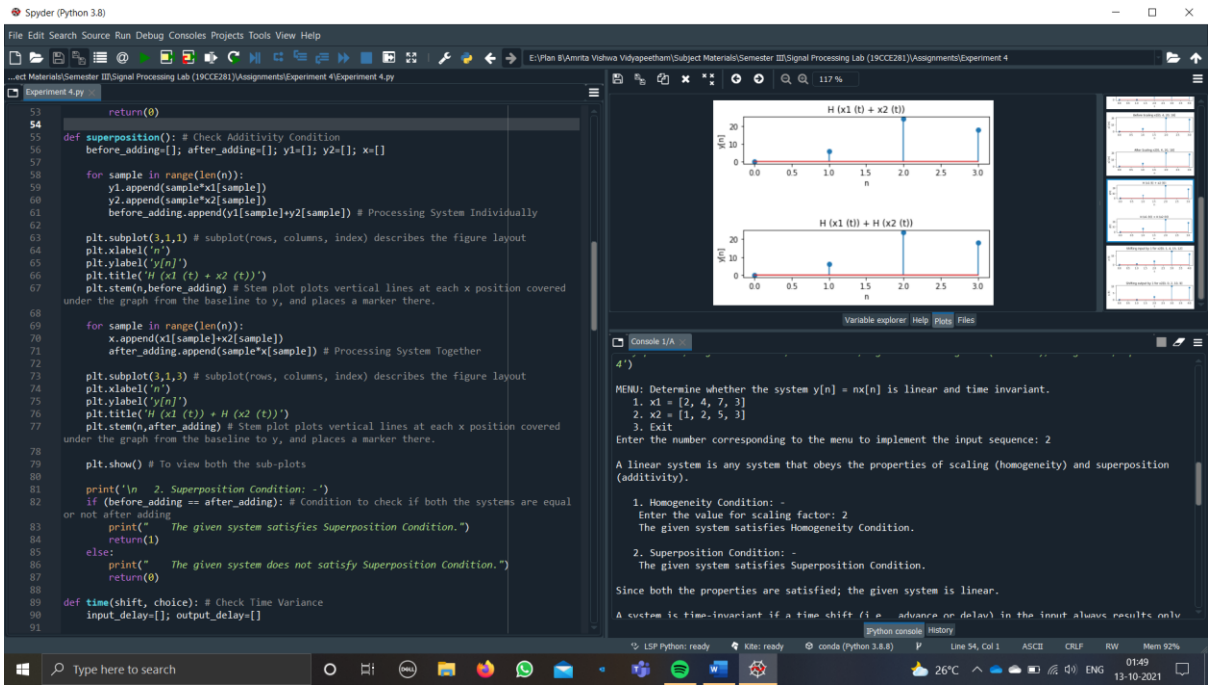




The given system satisfies the Homogeneity Condition.

Check Additivity Condition





Step 1: Process System Individually.

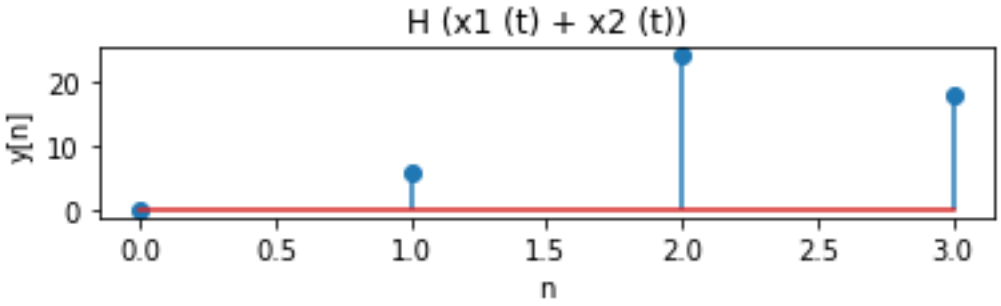
Step 2: Plot graph before adding.

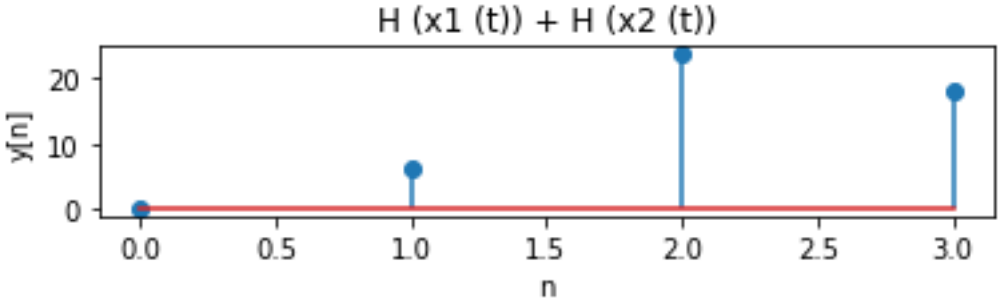
Step 3: Process System Together.

Step 4: Plot graph after adding.

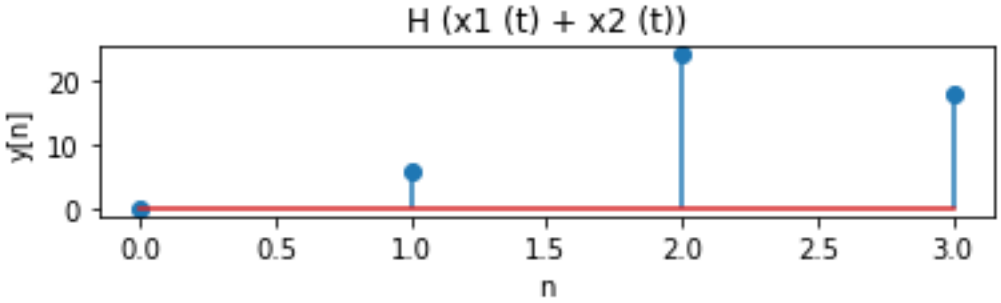
Step 5: Check if both the systems are equal or not after adding.

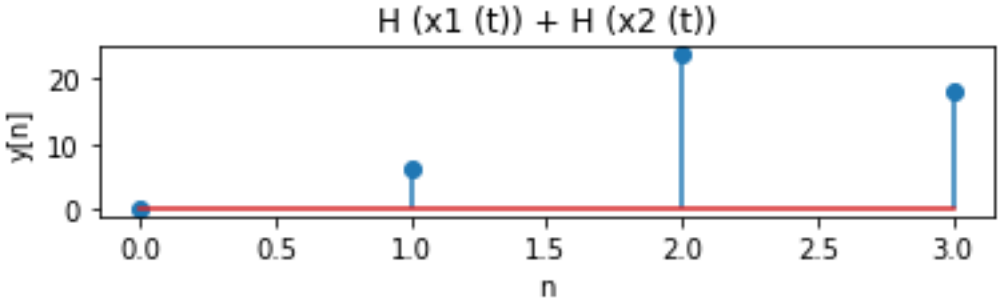
x1 = [2,4,7,3]





x2 = [1,2,5,3]

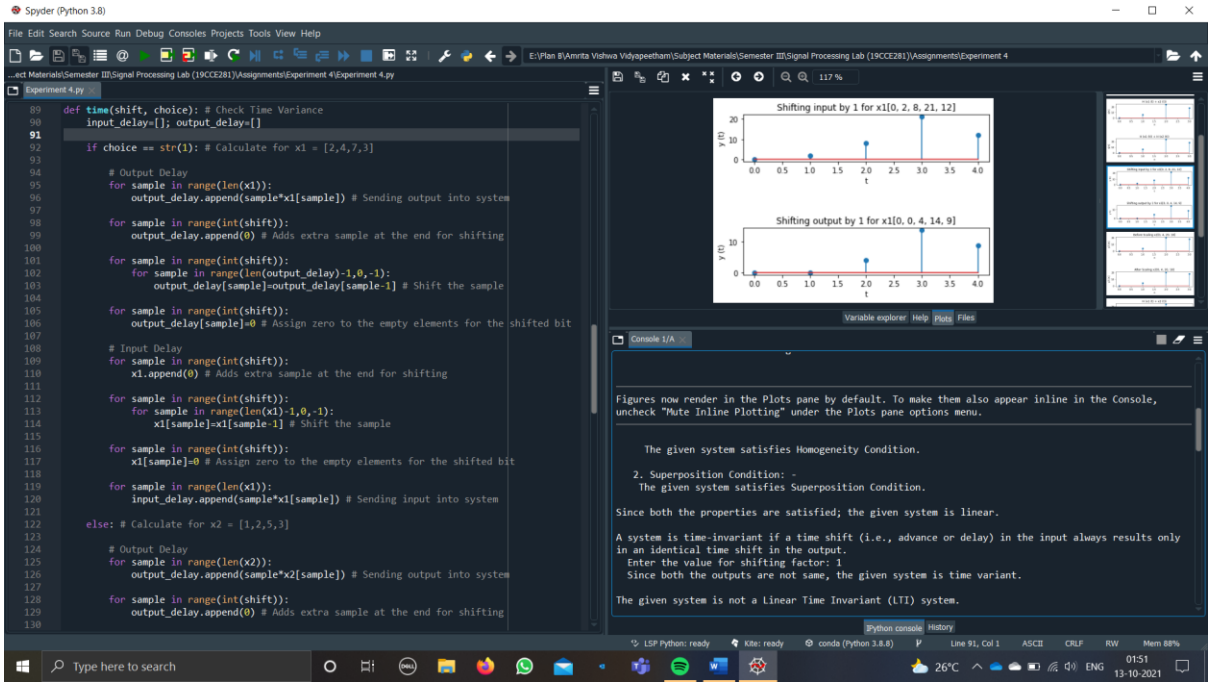


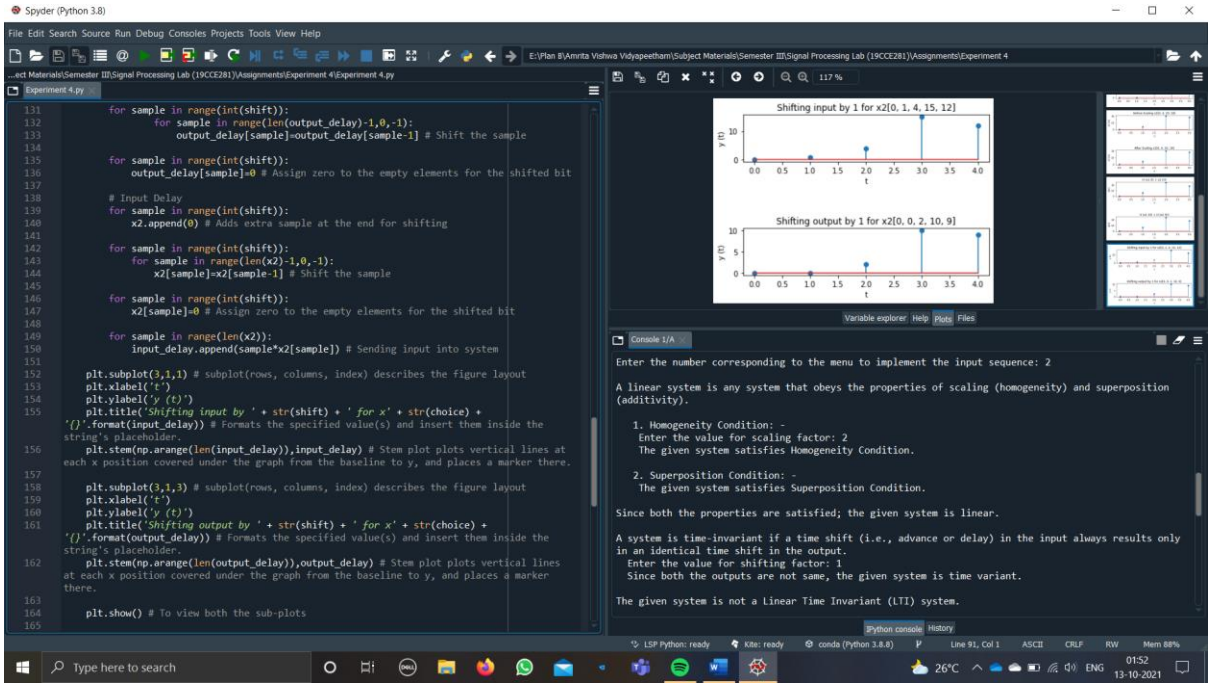


The given system satisfies Superposition Condition.

Since both the properties are satisfied; the given system is linear.

Check Time Variance



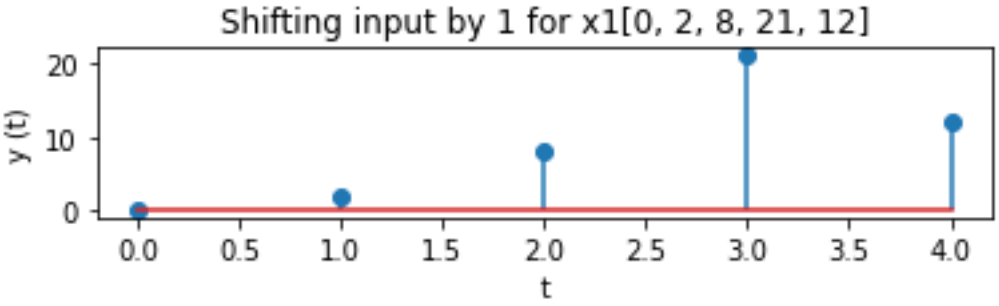


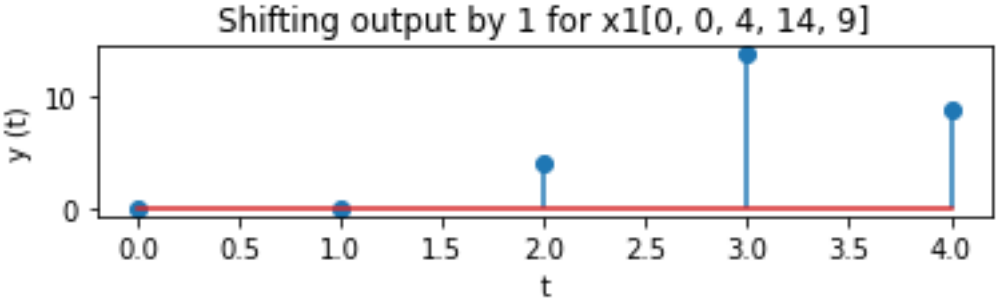
Step 1: Process input and output delay.

Step 2: Plot graphs before and after shifting.

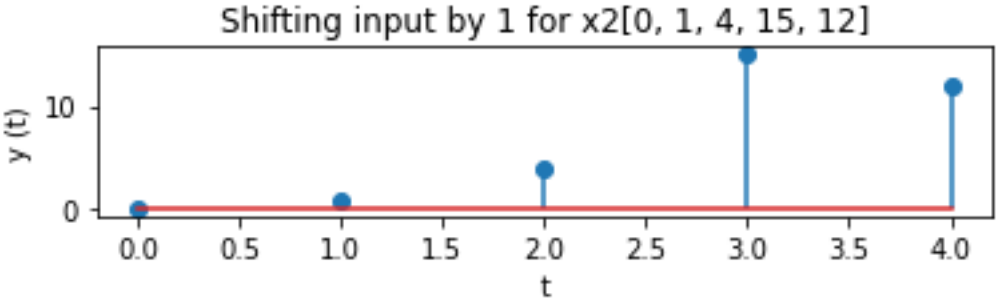
Step 3: Check if both the systems are equal or not after shifting.

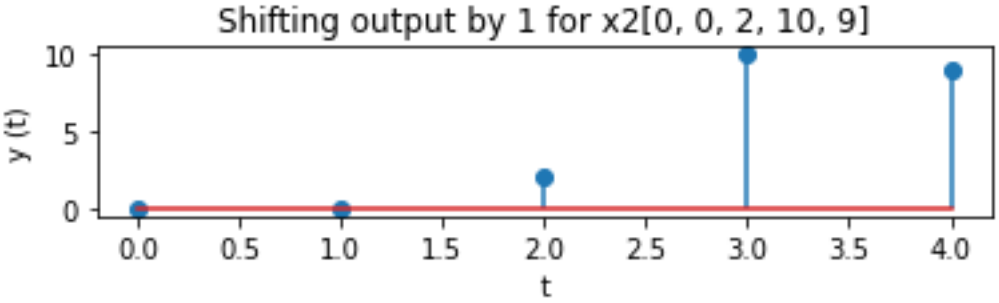
x1 = [2,4,7,3]





x2 = [1,2,5,3]

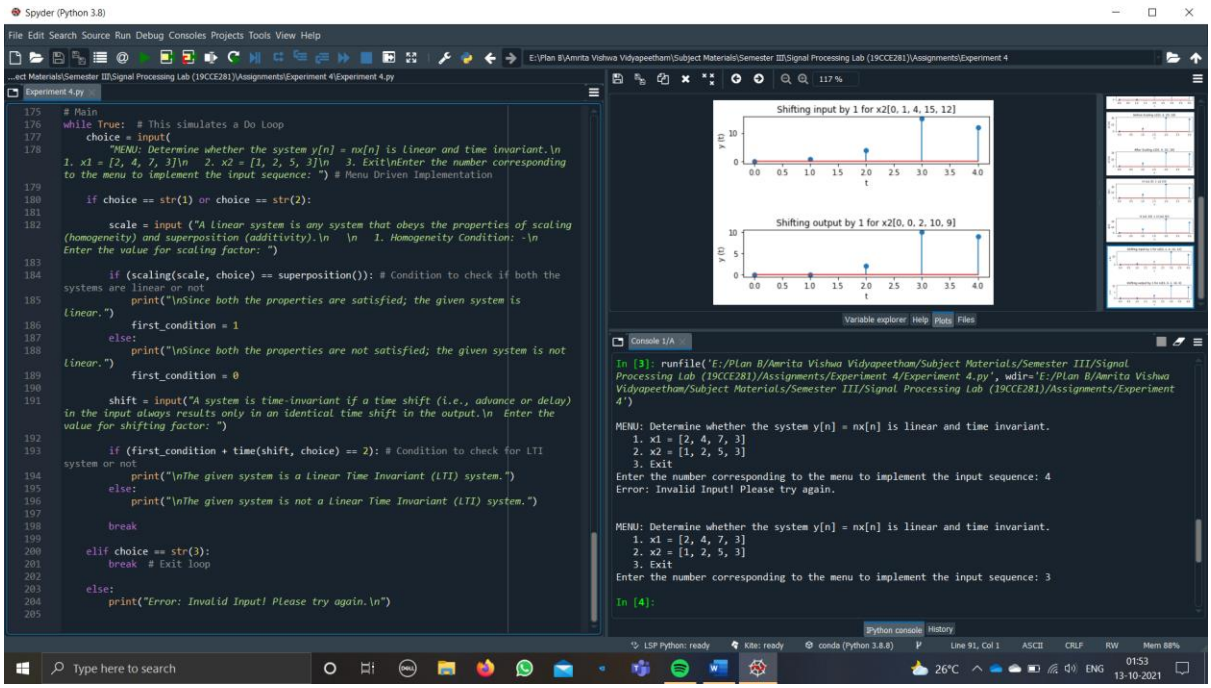




Since both the outputs are not the same, the given system is time-variant.

The given system is not a Linear Time-Invariant (LTI) system.

Main Menu-Driven Implementation



Thank You!

```
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

Restarting kernel...
```

```
In [1]: 'E:/Plan B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/
Signal Processing Lab (19CCE281)/Assignments/Experiment 4/Experiment 4.py'      ='E:/Plan
B/Amrita Vishwa Vidyapeetham/Subject Materials/Semester III/Signal Processing Lab
(19CCE281)/Assignments/Experiment 4'

MENU: Determine whether the system y[n] = nx[n] is linear and time invariant.
   1. x1 = [2, 4, 7, 3]
   2. x2 = [1, 2, 5, 3]
   3. Exit
Enter the number corresponding to the menu to implement the input sequence: 1

A linear system is any system that obeys the properties of scaling (homogeneity) and
superposition (additivity).

   1. Homogeneity Condition: -
    Enter the value for scaling factor: 2
    The given system satisfies Homogeneity Condition.

   2. Superposition Condition: -
    The given system satisfies Superposition Condition.

Since both the properties are satisfied; the given system is linear.

A system is time-invariant if a time shift (i.e., advance or delay) in the input always
results only in an identical time shift in the output.
  Enter the value for shifting factor: 1
  Since both the outputs are not same, the given system is time variant.

The given system is not a Linear Time Invariant (LTI) system.

In [2]:
```