CSE220 Signals and Linear Systems
Offline on Convolution

September 16, 2024

Version : 0.0

1 Introduction

In this offline, you will work with discrete signals, continuous signals, linear time invariant discrete and continuous systems. The goal of this offline is to help you understand and visualize how a linear time invariant system is completely characterized by its impulse response.

2 Discrete Signal

You will implement the class `DiscreteSignal`. It models a discrete-time signal. Each instance represents a discrete signal that can be manipulated, shifted, and combined with other signals.

Attributes:

- `values`: A numpy array of signal values.
- `INF`: A constant that determines the range of the signal values. The numpy array should hold signal values in range `(-INF, INF)`.

Methods:

- `__init__(self, INF)`: Initializes the signal.
- `set_value_at_time(self, time, value)`: Sets the value of the signal at a specific time index. The `time` parameter is the time index where the value should be set, and `value` is the new value to be assigned. This method modifies the signal in place and updates the signal array accordingly.
- `shift_signal(self, shift)`: Returns a new `DiscreteSignal` instance with the shifted signal $x[n - shift]$.

- `add(self, other)`: Returns a new `DiscreteSignal` instance representing the sum of the current signal and another signal.
- `multiply(self, other)`: Returns a new `DiscreteSignal` instance representing the element-wise multiplication of the current signal and another signal.
- `multiply_const_factor(self, scaler)`: Returns a new `DiscreteSignal` instance with the signal multiplied by a constant factor.
- `plot(self)`: Plots the signal.

You can add additional attributes, methods and parameters if you like.

3 Continuous Signal

You will implement the `ContinuousSignal` class. It models a continuous-time signal. Each instance represents a continuous signal that can be manipulated, shifted, added, and multiplied. The class uses a function to define the signal.

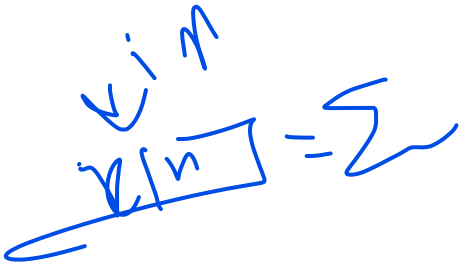
Attributes:

- `func`: A function representing the signal $x(t)$. The function takes a numpy array as input and returns the corresponding values of the signal for each element in the array as a numpy array.

Methods:

- `__init__(self, func)`: Initializes the signal with a given function.
- `shift(self, shift)`: Returns a new `ContinuousSignal` instance with the shifted signal $x(t - shift)$.
- `add(self, other)`: Returns a new `ContinuousSignal` instance representing the sum of the current signal and another signal.
- `multiply(self, other)`: Returns a new `ContinuousSignal` instance representing the multiplication of the current signal and another signal.
- `multiply_const_factor(self, scaler)`: Returns a new `ContinuousSignal` instance with the signal multiplied by a constant factor.
- `plot(self)`: Plots the signal.

You can add additional attributes, methods and parameters if you like.



4 Discrete Linear Time Invariant System

You will implement `LTI_Discrete` class. It represents an LTI system with a given impulse response.

Attributes:

- `impulse_response`: An instance of `DiscreteSignal` representing the system's impulse response.

Methods:

- `__init__(self, impulse_response)`: Initializes the LTI system with a given impulse response.
- `linear_combination_of_impulses(self, input_signal)`: Decomposes the input signal into a linear combination of unit impulses. Returns the unit impulses and their coefficients. Only the unit impulses having a non zero sample in the range (-INF, INF) are considered.
- `output(self, input_signal)`: Finds the output of the signal by first decomposing the input, then using the impulse response of the system. Returns the output signal, the system's response to the constituent impulses of the input signal and their coefficients.

You can add additional attributes, methods and parameters if you like.

5 Continuous Linear Time Invariant System

You will implement the class `LTI_Continuous`. It represents a Linear Time-Invariant (LTI) system with a given continuous-time impulse response.

Attributes:

- `impulse_response`: An instance of `ContinuousSignal` representing the system's impulse response.

Methods:

- `__init__(self, impulse_response)`: Initializes the LTI system with a given continuous-time impulse response.
- `linear_combination_of_impulses(self, input_signal, delta)`: Decomposes the input continuous signal into a linear combination of impulses of width `delta` and height `1/delta`. Returns the impulses and their coefficients.
- `output_approx(self, input_signal, delta)`: Finds the output of the signal by first decomposing the input, then using the impulse response of the system. Returns the output signal, the system's response to the constituent impulses of the input signal and their coefficients.

You can add additional attributes, methods and parameters if you like.

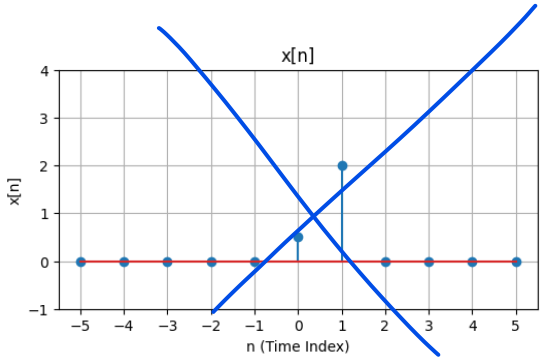


Figure 1: Input Discrete Signal, INF = 5

6 Main Function

Define a main function.

6.1 Discrete Portion

Create a discrete LTI with an arbitrary impulse response. Then create a discrete signal. Pass the signal to the `linear_combination_of_impulses(self, input_signal)` method. Plot the returned impulses multiplied by the coefficients and their sum in a subplot. Refer to figure 1 and 2.

Now call the `output(self, input_signal)` method. Plot the returned output and impulse responses multiplied by their corresponding coefficient. Refer to figure 3 and 4.

6.2 Continuous Portion

Similar to discrete portion, create a continuous LTI system and a continuous input signal. Pass the input signal to `linear_combination_of_impulses` method. Plot the returned impulses multiplied by the corresponding coefficients and the reconstructed signal by adding them together. See figure 5 and 6.

Also plot the reconstructed signal with varying delta from larger to smaller value. See figure 7.

Now call the `output_approx` method. Plot the returned output and impulse responses multiplied by their corresponding coefficient. Refer to figure 8.

Finally, determine the output by hand through integration. You can refer to your text book to find some examples of input signals and their corresponding output for a particular LTI. Then plot the output of the `output_approx` method against true output while varying delta. Refer to figure 9.

Save all the figures in disk. After running the main function, all the plots mentioned above should be saved in 2 folders, one for discrete and one for continuous system.

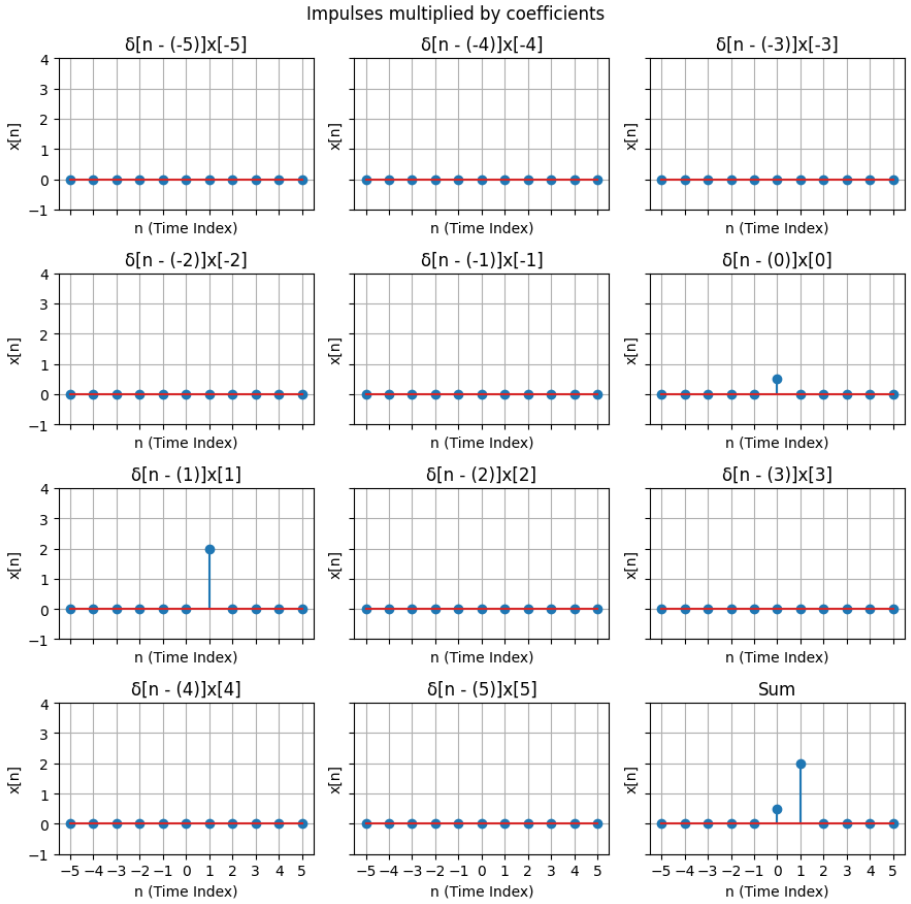


Figure 2: Returned impulses multiplied by respective coefficients

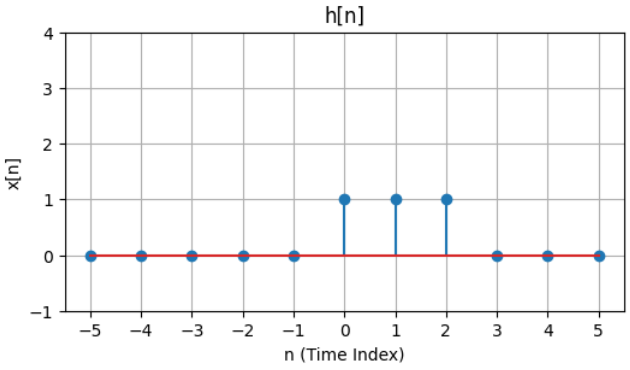


Figure 3: Impulse Response, INF = 5

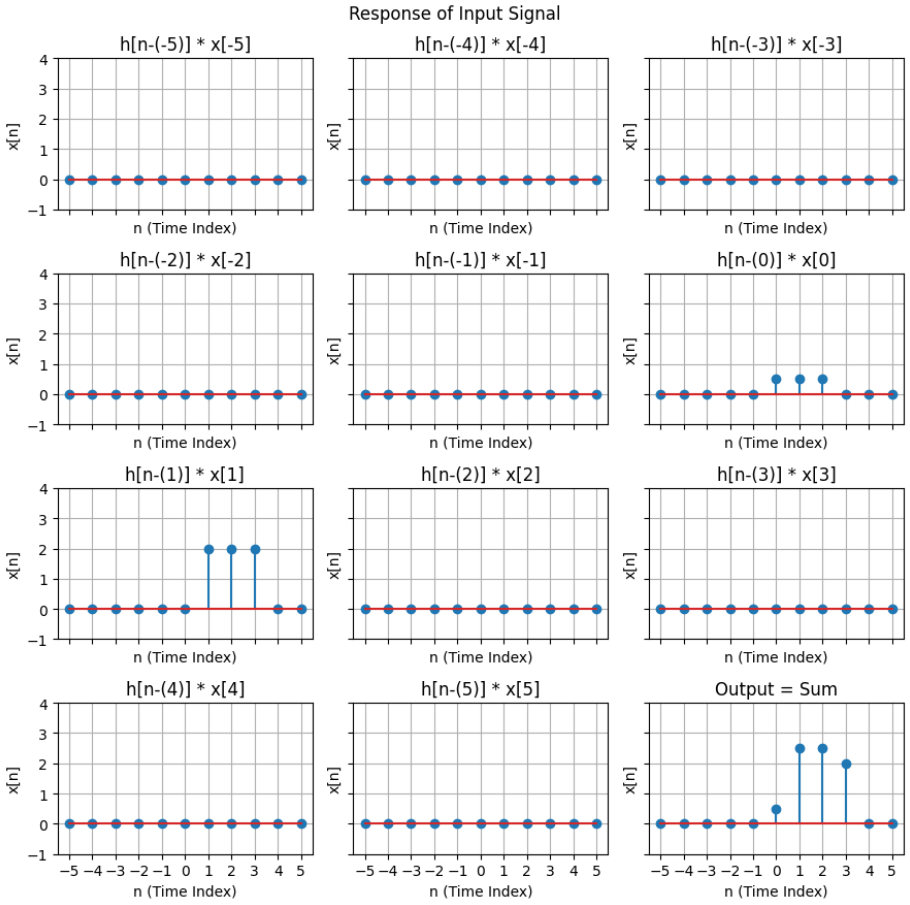


Figure 4: Output

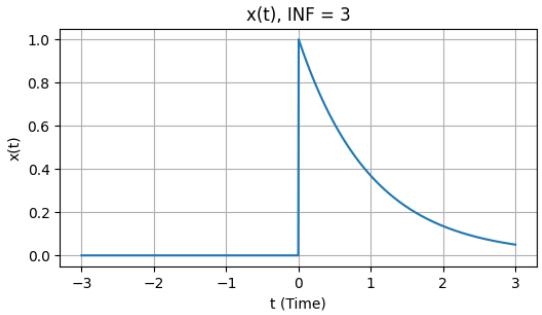


Figure 5: Input Signal

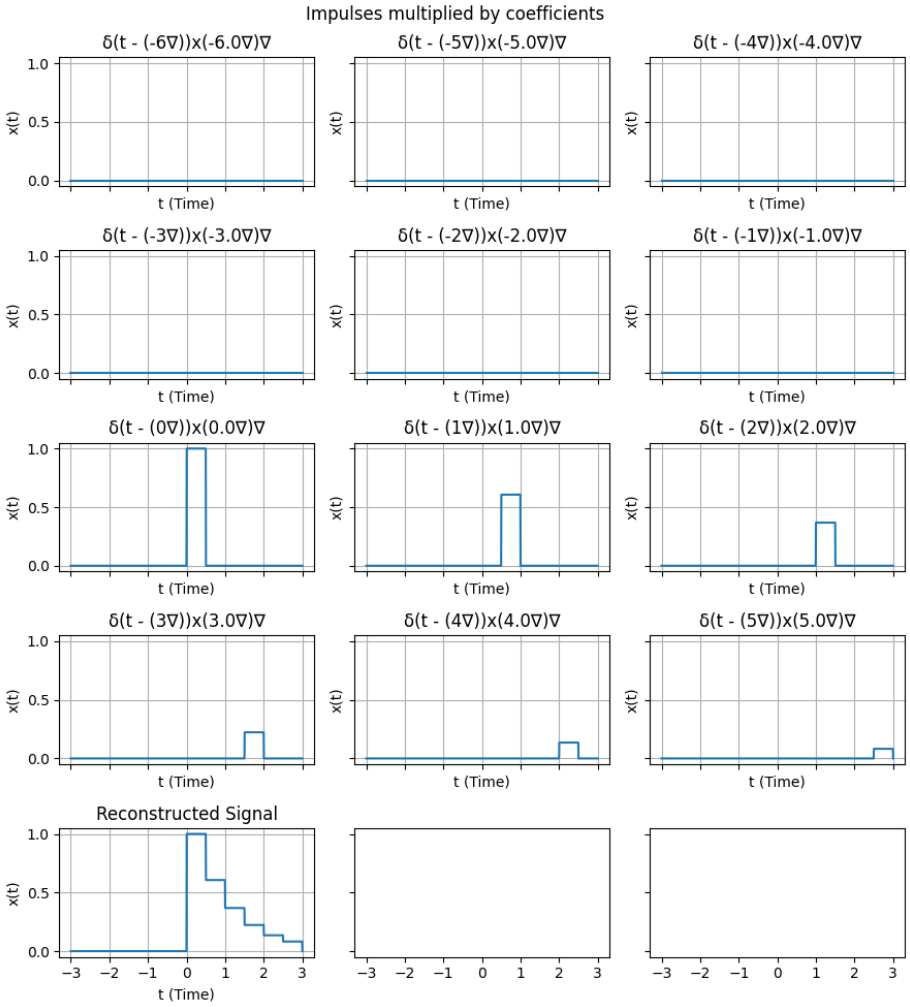


Figure 6: Returned impulses multiplied by their coefficients

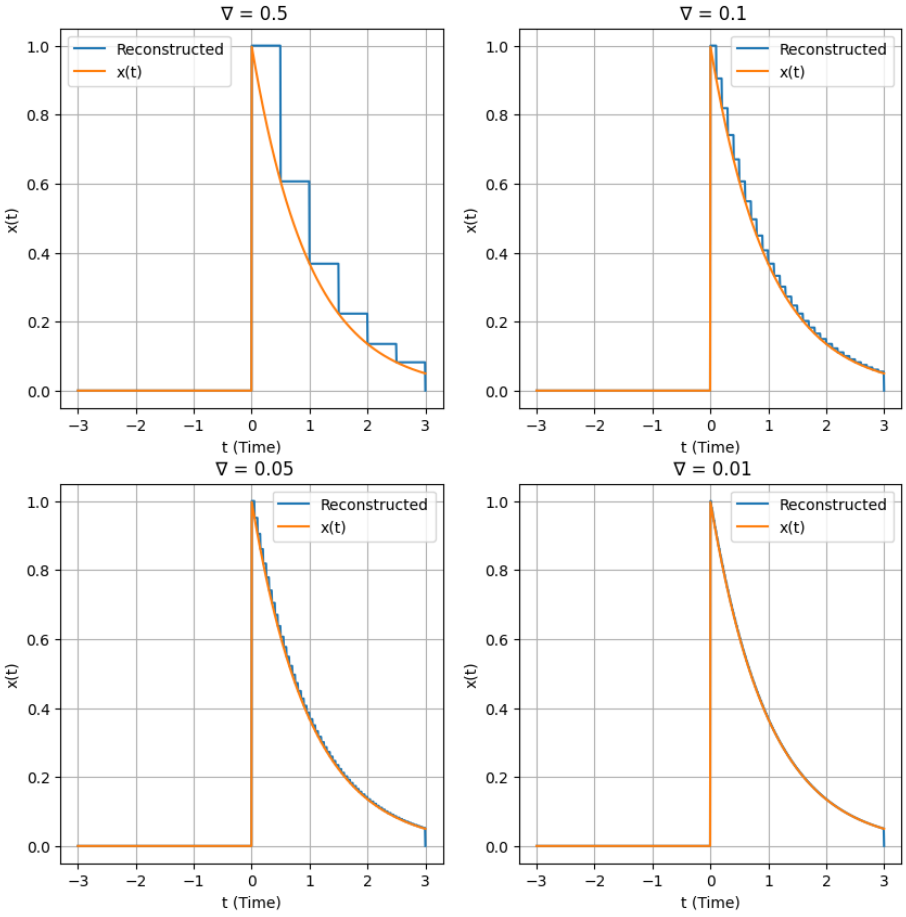


Figure 7: Reconstruction of input signal with varying delta

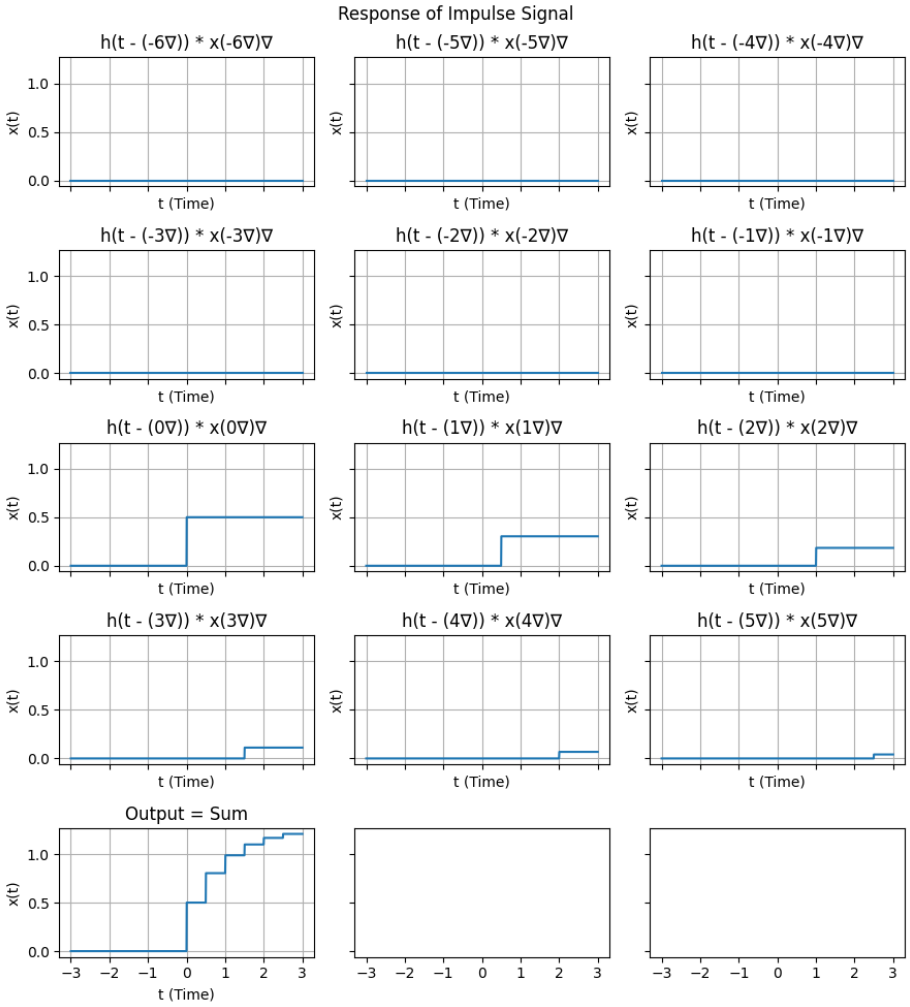


Figure 8: Returned impulses multiplied by their coefficients

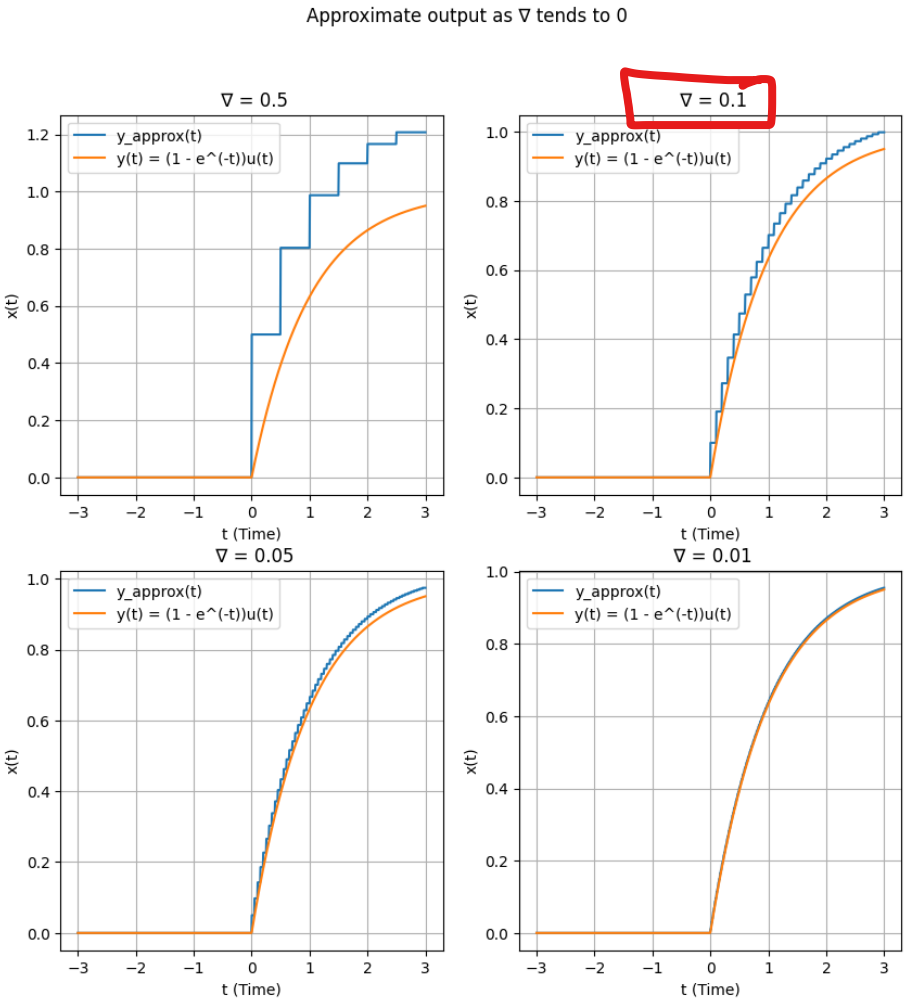


Figure 9: Returned impulses multiplied by their coefficients

7 Learning Resources

You can go through sections 2.1.1, 2.1.2, 2.2.1, 2.2.2 of the book SIGNALS and SYSTEMS second edition by ALAN V. OPPENHEIM, ALAN S. WILLSKY, WITH S. HAMID NAWAB

8 Mark Distribution

`DiscreteSignal` Class	20
`ContinuousSignal` Class	20
`LTI_Discrete` Class	20
`LTI_Continuous` Class	20
Plots and Main function	20

9 Submission

Put all of your code in a python file, rename it by your student id. Put the python file in a folder named by your student id. Zip the folder and submit the zip file. **Do not include any images in the folder**.
Submission Deadline: Friday, 27 September, 11:59 PM.