CSE220 Signals and Linear Systems
Practice on Continuous-Time Convolution (Impulse Decomposition)

1 Introduction

In this practice, you will implement **continuous-time signals and continuous-time LTI systems**. The goal is to help you understand and visualize how a continuous-time input signal $x(t)$ can be approximated as a **linear combination of narrow impulses**. This is the key idea behind the continuous-time convolution integral:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)\, h(t-\tau)\, d\tau.$$

Here, we will only focus on the **input decomposition step** (and not the full output computation).

2 Continuous Signal

You will implement the `ContinuousSignal` class. It models a continuous-time signal. Each instance represents a continuous-time signal that can be manipulated, shifted, added, and multiplied. The class uses a Python 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 signal values 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 $x(t - \mathit{shift})$.
- **add(self, other):** Returns a new `ContinuousSignal` instance representing $x(t) + y(t)$.
- **multiply(self, other):** Returns a new `ContinuousSignal` instance representing $x(t)\, y(t)$.
- **multiply_const_factor(self, scaler):** Returns a new `ContinuousSignal` instance representing $a\, x(t)$.
- **plot(self, t_min, t_max, num_points, title=...):** Plots the signal over the given time range.

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

3 Continuous Linear Time Invariant System (Partial)

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

Attributes:

- **impulse_response:** An instance of `ContinuousSignal` representing the system's impulse response $h(t)$.

Methods:

- **__init__(self, impulse_response):** Initializes the LTI system with a given impulse response.
- **linear_combination_of_impulses(self, input_signal, delta):** Decomposes the continuous-time input signal into a linear combination of **rectangular impulses** of width Δ and height $1/\Delta$. Returns the impulses and their coefficients.
- **output_approx(self, input_signal, delta): Not required in this practice.** Keep it as a stub (`pass` or `raise NotImplementedError`).

Impulse approximation model to use

Define the rectangular impulse approximation:

$$\delta_\Delta(t) = \begin{cases} \frac{1}{\Delta}, & 0 \le t < \Delta \\ 0, & \text{otherwise} \end{cases}$$

and shifted versions $\delta_\Delta(t - t_k)$ where $t_k = k\Delta$.

Approximate the input signal using:

$$x(t) \approx \sum_k x(t_k)\Delta \; \delta_\Delta(t - t_k).$$

Therefore, your method should compute:

$$c_k = x(t_k)\Delta \quad \text{and} \quad \text{impulse}_k(t) = \delta_\Delta(t - t_k),$$

and return $\{(\text{impulse}_k, c_k)\}$ (as two lists or any convenient structure).

4 Main Function

Define a `main()` function and complete the following steps.

4.1 Step 1: Create a continuous LTI system and a continuous input signal

Choose a time window $[-T, T]$ for visualization (e.g., $T = 3$). Define:

- An input signal $x(t)$. Suggested: $x(t) = e^{-t}u(t)$, where $u(t)$ is the unit step.
- An impulse response $h(t)$. Suggested: $h(t) = u(t)$. (You will not use it for output in this practice, but include it to build the system.)

4.2 Step 2: Decompose the input using linear combination of impulses

Call:

$$(\{\text{impulse}_k\}, \{c_k\}) = \texttt{linear\_combination\_of\_impulses}(x, \Delta).$$

Then form each component signal:

$$x_k(t) = c_k \, \text{impulse}_k(t).$$

Finally reconstruct:

$$\hat{x}(t) = \sum_k x_k(t).$$

4.3 Step 3: Required plots

You must generate and save the following figures.

Figure 1: Input Signal
Plot $x(t)$ over $[-T, T]$.

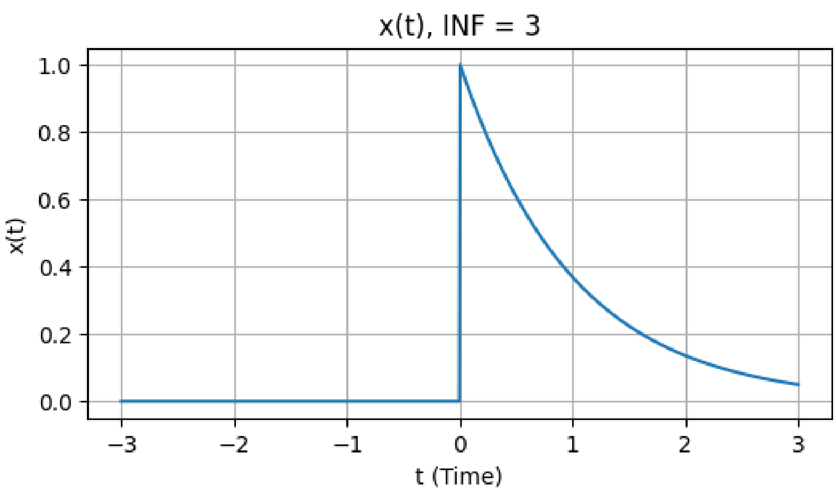

Figure 1: Input Signal

Figure 2: Returned impulses multiplied by their coefficients, and reconstruction
Create a grid of subplots. Plot several component signals $x_k(t) = c_k \, \delta_\Delta(t - t_k)$ (including zero components if they appear), and include one subplot that shows the reconstructed signal $\hat{x}(t)$.

Figure 3: Reconstruction with varying Δ
Choose multiple values of Δ (e.g., $0.5, 0.1, 0.05, 0.01$). For each Δ, overlay plots of $x(t)$ and $\hat{x}(t)$ to show that $\hat{x}(t)$ approaches $x(t)$ as Δ becomes smaller.

Saving requirement: After running `main()`, all plots must be saved in a folder named `continuous_practice/`. Do **not** submit images.

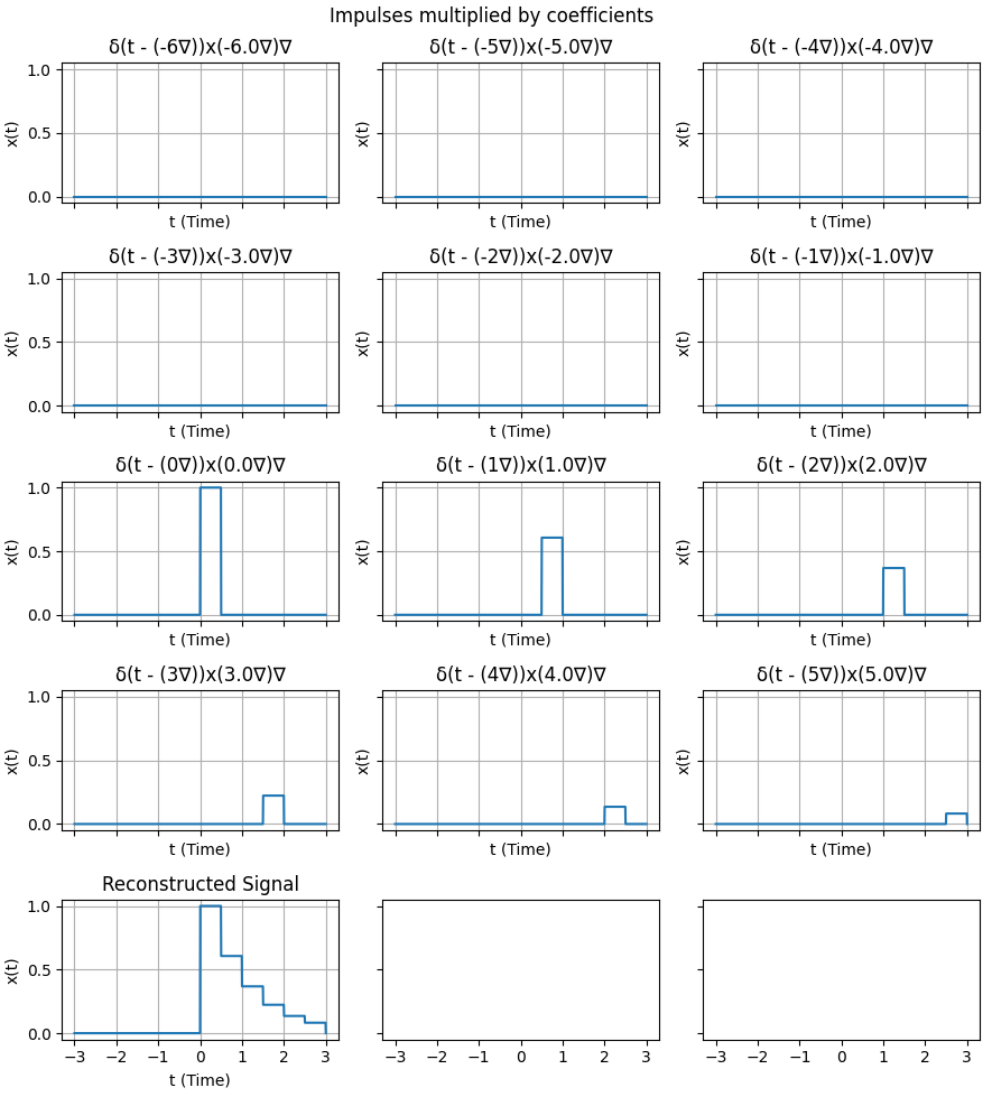

Figure 2: Returned impulses multiplied by their coefficients and reconstructed signal

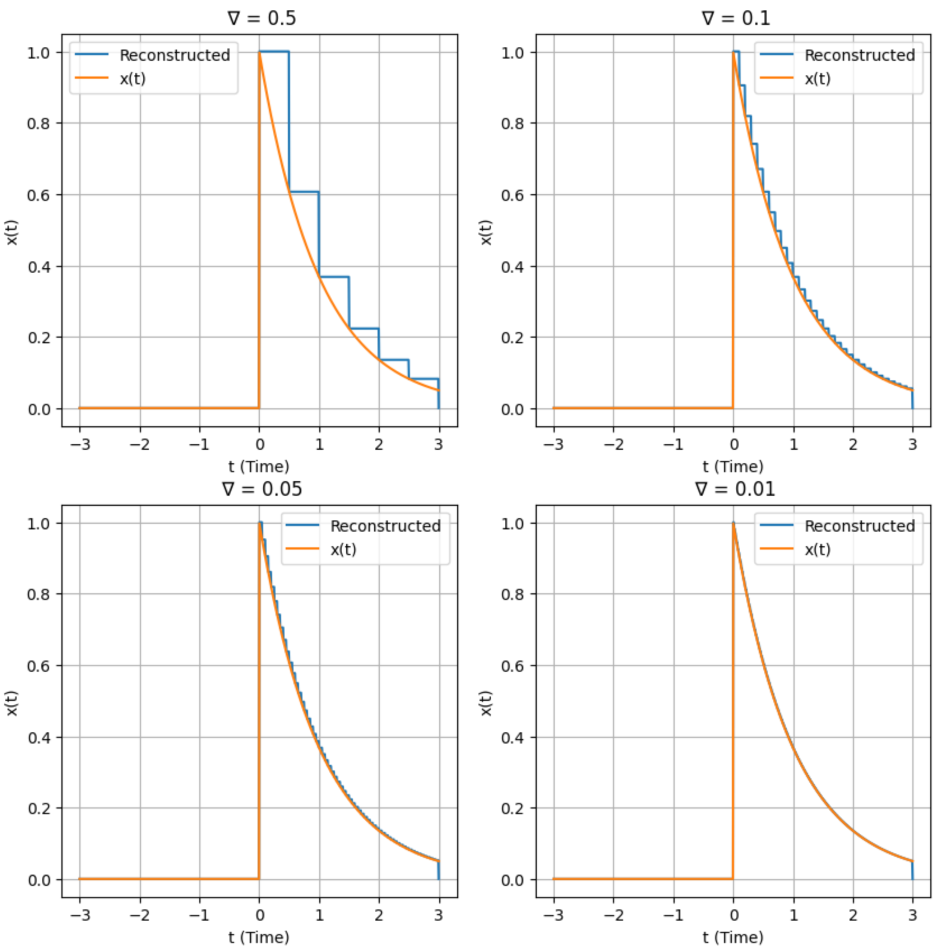

Figure 3: Reconstruction of input signal with varying Δ