July 2025 CSE220: Signals and Linear Systems Sessional

Offline Assignment on Continuous Fourier Transform

General Instructions

- All implementations must use **Object-Oriented Programming**.
- Use of any built-in Fourier Transform functions is **strictly prohibited**, including:
  - `np.fft`, `fft2`, `ifft`, `scipy.fft`
- All continuous integrals must be approximated numerically using `np.trapz()` or `np.trapezoid()` only.
- Complex exponentials must be handled by explicitly separating **real** and **imaginary** parts.
- Hard-coded Fourier transform pairs are forbidden.
- Skeleton codes (i.e., `task1_signal_framework.py` and `task2_image_cft.py`) will be provided with this assignment and must not be modified structurally.
- You are not allowed to initialize/declare any other new classes or new attributes/methods inside a given class. Follow the convention provided in the problem statement.

1 Task 1: Object-Oriented Continuous Signal Modeling and Spectral Analysis

1.1 Objective

The objective of this task is to design a reusable and extensible **object-oriented framework** for modeling continuous-time signals, computing their **Continuous Fourier Transform (CFT)**, and reconstructing them using the **Inverse Continuous Fourier Transform (ICFT)**.

Rather than analyzing a single predefined signal, students are required to build a general-purpose signal analysis system capable of generating, combining, and analyzing a wide variety of continuous-time signals.

1.2 Background

In practical signal processing systems, signals are often composed of multiple components including periodic oscillations, polynomial trends, and time-limited pulses. Such signals cannot be analyzed effectively using rigid, hard-coded formulations.

This task introduces abstraction through classes, allowing signals to be:

- Parameterized
- Composed dynamically
- Analyzed in a unified framework

1.3 Definition and Generation of Signal Types

In this assignment, continuous-time signals are generated and analyzed using an **object-oriented framework**. The `SignalGenerator` class defines various signal types as methods, each returning the signal values over a continuous time axis. The problem statement is to implement and combine these signals to form complex waveforms without using hard-coded expressions.

1. Periodic Signals

Periodic signals repeat after a fixed period $T = \frac{1}{f}$. They are defined mathematically as:

$$x(t) = \begin{cases} A\sin(2\pi ft) & \text{(sine wave)} \\ A\cos(2\pi ft) & \text{(cosine wave)} \end{cases}$$

2. Non-Sinusoidal Periodic Signals

These signals are periodic but non-sinusoidal. They can be constructed mathematically using functions like sign, floor, and absolute value:

$$\begin{aligned} \text{Square wave: } & x(t) = A\,\text{sign}(\sin(2\pi ft)) \\ \text{Sawtooth wave: } & x(t) = A\left[2\left(ft - \lfloor 0.5 + ft \rfloor\right)\right] \\ \text{Triangle wave: } & x(t) = \frac{2A}{\pi}\sin^{-1}(\sin(2\pi ft)) \end{aligned}$$

3. Polynomial Signals

Polynomial signals vary smoothly with time according to a polynomial function:

$$x(t) = ct^2 \quad \text{(parabolic)}, \quad x(t) = ct^3 \quad \text{(cubic)}$$

4. Time-Limited Signals

Time-limited signals exist only within a specific time interval and are zero elsewhere:

$$x(t) = \begin{cases} 1, & |t| \le \frac{\text{width}}{2} \quad \text{(rectangular)} \\ 1, & \text{start} \le t \le \text{end} \quad \text{(pulse)} \\ 0, & \text{otherwise} \end{cases}$$

Each signal must be defined over a continuous time axis and evaluated numerically.

1.4 Composite Signal Requirement

Using the developed framework, construct the following composite signal:

$$x(t) = 2\sin(2\pi t) + 0.5\cos(6\pi t) + \text{Square}(t) + t^3 \cdot \text{Rect}(t)$$

Important: This expression must not be implemented directly. Each component must be created as a separate object and combined programmatically using the composite signal class.

1.5 Class Specifications

1. `ContinuousSignal` (Abstract Base Class)

Purpose: Defines a common interface for all continuous-time signals.

Attributes:

- `t`: Continuous time axis

Methods:

- `values()` – returns signal samples
- `plot()` – visualizes the signal in time domain

2. `SignalGenerator` (Inherits `ContinuousSignal`)

Purpose: Generates parameterized continuous-time signals defined in Section 1.3.

Methods:

- `sine(amplitude, frequency)`
- `cosine(amplitude, frequency)`
- `square(amplitude, frequency)`
- `sawtooth(amplitude, frequency)`
- `triangle(amplitude, frequency)`
- `cubic(coefficient)`
- `parabolic(coefficient)`
- `rectangular(width)`
- `pulse(start, end)`

3. `CompositeSignal` **(Inherits** `ContinuousSignal`**)**

Purpose: Combines multiple signal components into a single signal.

Attributes:

- `components`: list of signal objects

Methods:

- `add_component(signal)`
- `values()`

4. `CFTAnalyzer`

Purpose: Computes the Continuous Fourier Transform.

Attributes:

- `signal`: CompositeSignal
- `t`: Continuous time axis
- `frequencies`: Continuous frequency axis

Methods:

- `compute_cft()`: Computes frequency domain components of the CFT
- `plot_spectrum()`

5. `InverseCFT`

Purpose: Reconstructs time-domain signals using ICFT.

Attributes:

- `spectrum`: Tuple consisting two numpy arrays indicating frequency-domain components
- `frequencies`: Continuous frequency axis
- `t`: Continuous time axis

Methods:

- `reconstruct()`

1.6 Analysis Requirements

Students must:

- Compare original and reconstructed signals
- Study reconstruction under different frequency truncation limits
- Comment on numerical errors and Gibbs phenomenon

2 Task 2: Two-Dimensional Continuous Fourier Transform for Image Processing

Objective

The objective of this task is to design and implement a complete object-oriented framework for performing the **Two-Dimensional Continuous Fourier Transform (2D-CFT)** and its inverse on grayscale images, using only numerical integration techniques. This task emphasizes the theoretical foundations of frequency-domain image analysis and demonstrates how spatial-domain image characteristics map into the frequency domain.

Students are strictly prohibited from using FFT, DFT, or any built-in frequency transform utilities. All transformations must be computed using numerical integration (e.g., trapezoidal rule).

Theoretical Background

Let $I(x, y)$ represent a continuous grayscale image defined over spatial coordinates (x, y). The **2D Continuous Fourier Transform** of $I(x, y)$ is defined as:

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(x, y)\, e^{-j2\pi(ux+vy)}\, dx\, dy$$

Using Euler's identity, this can be expressed as:

$$\Re\{F(u, v)\} = \int \int I(x, y) \cos\left(2\pi(ux + vy)\right)\, dx\, dy$$

$$\Im\{F(u, v)\} = -\int \int I(x, y) \sin\left(2\pi(ux + vy)\right)\, dx\, dy$$

The inverse 2D-CFT is defined as:

$$I(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)\, e^{j2\pi(ux+vy)}\, du\, dv$$

This operation reconstructs the spatial image from its frequency-domain representation.

Task Description

You are required to perform the following steps:

1. Load and normalize a grayscale image and interpret it as a continuous 2D signal. (**Done for you**)
2. Compute the 2D Continuous Fourier Transform using separable numerical integration.
3. Visualize the magnitude spectrum of the transformed image.
4. Design and apply a frequency-domain low-pass filter. (**Done for you**)
5. Reconstruct the filtered image using the inverse 2D-CFT.

Object-Oriented Design Specifications

Your implementation must follow the class structure described below.

Class: `ContinuousImage`

Purpose: Represents a grayscale image as a continuous spatial signal.
Attributes:

- `image`: A normalized 2D array representing image intensity.
- `x`: Continuous spatial coordinate vector along the horizontal axis.
- `y`: Continuous spatial coordinate vector along the vertical axis.

Methods:

- `show(title)`: Displays the image with appropriate scaling and labels.

Class: `CFT2D`

Purpose: Computes the 2D Continuous Fourier Transform of a given image using numerical integration.
Attributes:

- `I`: Input image intensity matrix.
- `x`, `y`: Spatial coordinate grids.

Methods:

- `compute_cft()`:
  - Computes real and imaginary components of the 2D-CFT
  - Uses separable trapezoidal integration
- `plot_magnitude()`:
  - Visualizes the logarithmic magnitude spectrum

Class: `FrequencyFilter` **(Done for you)**

Purpose: Applies frequency-domain filtering operations.
Methods:

- `low_pass(real, imag, cutoff)`:
  - Suppresses high-frequency components beyond a cutoff radius
  - Preserves low-frequency image information

Class: `InverseCFT2D`

Purpose: Reconstructs the spatial-domain image using inverse 2D-CFT.
Attributes:

- `real`, `imag`: Filtered frequency-domain components.
- `x`, `y`: Frequency coordinate vectors.

Methods:

- `reconstruct()`:
  - Performs inverse 2D numerical integration
  - Produces reconstructed image

Constraints and Rules

- Use of FFT, DFT, or frequency-domain libraries is strictly forbidden.
- All integrations must be implemented using numerical integration (e.g., trapezoidal rule).
- Object-oriented design must be strictly followed.
- Code must be well-documented and modular.

Marks Distribution

Class	Mark
SignalGenerator	20
CFTAnalyzer	20
InverseCFT	20
CFT2D	20
InverseCFT2D	20
Total	**100**

Submission

Create separate python files for the two parts. Rename them as id_first.py and id_second.py. Put the python files in a folder named by your student id. Zip the folder and submit the zip file. Do not include any images in the folder. Deadline: Monday, 9 February, 11:59 PM.