Exercise 1 — DFT, Sampling, and Aliasing

Tools

Online notebook: `https://colab.research.google.com/`

Coding in **Python** (NumPy + Matplotlib).

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Board / Paper Tasks

Example 1: Hand DFT

Compute the 4-point DFT $F[k]$ of $f[n] = [1,\ 2,\ 0,\ 0]$ for $k = 0, 1, 2, 3$ using

$$F[k] = \sum_{n=0}^{3} f[n]\, e^{-j2\pi kn/4}.$$

Then compute IDFT to recover $f[n]$.

Code:

```
f = np.array([1.,2.,0.,0.])
F = np.fft.fft(f)
print(F)

f2 = np.fft.ifft(F)
print(f)
```

Python Tasks (run and discuss)

Example 2

Generate a (simulated) continuous-time signal $s(t) = \sin(2\pi f_0 t)$ with a sufficiently high frequency f_0. Sample the signal with sampling frequency $f_s = 200$

Visualize the signal in the time domain.

```
plt.plot(t,s)
```

Compute and plot its magnitude spectrum using the FFT.

```
np.abs(np.fft.fft(s))

freqs = np.fft.fftfreq(len(t), 1/fs)
plt.plot(freqs,np.abs(S))
```

Consider and compare the following cases:

$$f_0 = 0.2 f_s \quad \text{(below Nyquist limit)}$$
$$f_0 = 0.6 f_s \quad \text{(above Nyquist limit)}$$

- How does the sampled waveform change when $f_0 > f_s/2$?
- How is the aliased (apparent) frequency related to the original one?
- How does this illustrate the Nyquist–Shannon sampling theorem?

Example 3

Repeat Example 2 with $(f_s, f_0) \in \{(20, 15), (24, 15), (30, 15)\}$. For each case:

- Plot time-domain samples (markers) over the underlying continuous curve.
- Plot the magnitude spectrum.
- Predict f_a analytically and verify from the spectrum.

Example 4

1. Load a grayscale image (you may use any small image).
   ```
   img = Image.open("fin.png") #change the path
   f = img.convert("L") #convert to grayscale
   ```
2. Compute the 2D Discrete Fourier Transform (DFT) using:
   ```
   F = np.fft.fft2(f)
   ```
3. Visualize the magnitude spectrum:
   ```
   plt.imshow(np.log(1 + np.abs(F)), cmap='gray')
   ```
4. Apply the frequency shift to move the zero-frequency component to the center:
   ```
   Fshift = np.fft.fftshift(F)
   plt.imshow(np.log(1 + np.abs(Fshift)), cmap='gray')
   ```
5. Reconstruct the image by applying the inverse transform:
   ```
   img_rec = np.fft.ifft2(F)
   plt.imshow(np.real(img_rec), cmap='gray')
   ```
6. Compare the reconstructed image with the original one. Verify that the inverse transform perfectly restores the image (up to numerical precision).

Example 5

1. Generate synthetic pattern

```
N = 512  # size of the "continuous" reference grid
f_x = 0.25   # spatial frequency in x-direction [cycles per
   pixel]
f_y = 0.05   # spatial frequency in y-direction [cycles per
   pixel]

x = np.arange(N)
y = np.arange(N)
X, Y = np.meshgrid(x, y)
img_cont = np.sin(2 * np.pi * (f_x * X + f_y * Y))
```

2. Sample this continuous pattern at different sampling steps to simulate different spatial resolutions.

```
# Downsample
img_sub = img_cont[::step, ::step]

# Upsample back for visualization
img_resized = np.kron(img_sub, np.ones((step, step)))
img_resized = img_resized[:N, :N]  # crop to original size
```

3. Visualize the resulting images and observe how aliasing appears when the sampling step increases.

```
plt.imshow(np.log(1 + np.abs(F)), cmap='gray')
```

4. Optionally, compare with the same experiment on a real image.

Example 6

Use $f_s = 200$ and $f_0 = 7.5$ (non-integer period). Compare number of samples $N \in \{64, 128, 512\}$ (length of the signal):

```
N_list = [64, 128, 512]
fs = 200
f0 = 7.5
for i,N in enumerate(N_list,1):
    t = np.arange(N)/fs
    x = np.sin(2*np.pi*f0*t)

    X = np.fft.fft(x); f = np.fft.fftfreq(len(x), 1/fs)
    plt.subplot(3,1,i)
    plt.stem(f, np.abs(X)/np.max(np.abs(X)))
    plt.xlim(0, 50); plt.ylim(0, 1.05); plt.grid(True); plt.title
       (f'N={N}, f0={f0} Hz')
    plt.xlabel('Frequency [Hz]')
    plt.ylabel('Normalized |X|')
    plt.tight_layout(); plt.show()
```

Discuss peak width vs. N. What visual evidence of leakage do you see?

Example 7

To demonstrate spectral leakage in images by comparing the Fourier spectra of a full image and of a cropped (windowed) region.

1. Generate a 2D sinusoidal pattern $f(x, y) = \sin(2\pi(f_x x + f_y y))$, where f_x and f_y are spatial frequencies (in cycles per pixel). Choose such values that the pattern does not fit an integer number of periods in the image window.

2. Compute and visualize its 2D Fourier magnitude spectrum

3. Select a smaller rectangular region of the image (for example, the top-left 128×128 pixels) and compute its spectrum as well

```
img = np.array(img)
crop_size = 128
img_crop = img[:crop_size, :crop_size]
```

4. Compare both spectra. Observe how sharp cropping in the spatial domain introduces energy spreading (leakage) in the frequency domain.

Example 8

Apply a 2D Hanning window to the cropped image from Example 7 before computing the FFT and compare all spectra.

```
hann_x = np.hanning(crop_size)
hann_y = np.hanning(crop_size)
window_2d = np.outer(hann_x, hann_y)
img_windowed = img_crop * window_2d
F_win = np.fft.fftshift(np.fft.fft2(img_windowed))
F_win_mag = np.log(1 + np.abs(F_win))
```