Exercise 1 — solution

Board / Paper Tasks

Example 1: Hand DFT

$F(k) = [3; 1 - 2i; -1; 1 + 2i]$

Python Tasks (run and discuss)

Example 2

```
fs = 200 #sampling frequency
T = 1
t   = np.arange(0, T, 1/fs)
f0 = 0.2*fs
print(f0)

s = np.sin(2*np.pi*f0*t)
S = np.fft.fft(s)

freqs = np.fft.fftfreq(len(t), 1/fs)

plt.figure(figsize=(10,3))
plt.subplot(1,2,1)

# "Continuous" reference signal (simulated with finer time step)
t_cont = np.linspace(0, T, 5000)
s_cont = np.sin(2*np.pi*f0*t_cont)
plt.plot(t_cont, s_cont, 'lightgray', label='Continuous signal')
    # smooth reference curve

# Sampled signal
s = np.sin(2*np.pi*f0*t)
plt.plot(t, s, 'o', color='C0', label='Samples')  # discrete
   samples as dots

plt.subplot(1,2,2)
plt.plot(freqs,np.abs(S),'o')
plt.show()
```

- 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***?

Case 1: $f_0 = 0.2f_s$ **(below Nyquist limit)** When the sampling frequency f_s is more than twice the signal frequency ($f_s > 2f_0$), the Nyquist condition is satisfied. The samples correctly capture the oscillations of the original signal. The discrete samples follow the true sinusoid, and the spectrum shows two distinct peaks at frequencies $\pm f_0$. No distortion or ambiguity occurs.

Case 2: $f_0 = 0.6f_s$ **(above Nyquist limit → aliasing)** Here, the sampling rate is too low – the Nyquist condition is violated. As a result, the samples of the high-frequency sinusoid appear as if they came from a lower-frequency signal. This effect is called ***aliasing***.

The apparent (aliased) frequency f_a is given by

$$f_a = |f_0 - kf_s|,$$

where k is an integer chosen so that f_a lies in the range $[0, f_s/2]$. In this example, $f_0 = 0.6f_s$ gives $f_a = 0.4f_s$. Therefore, the sampled signal seems to oscillate slower than the true one.

In the spectrum, the energy originally at frequency f_0 is folded (reflected) back into the lower-frequency region. The FFT magnitude plot shows a peak at the alias frequency f_a instead of f_0.

Example 3

```
for fs in [20, 24, 30]:
  T = 1
  t  = np.arange(0, T, 1/fs)
  f0 = 15
  print(fs)

  s = np.sin(2*np.pi*f0*t)
  S = np.fft.fft(s)

  freqs = np.fft.fftfreq(len(t), 1/fs)

  plt.figure(figsize=(10,3))
  plt.subplot(1,2,1)

  # "Continuous" reference signal (simulated with finer time step
     )
  t_cont = np.linspace(0, T, 5000)
  s_cont = np.sin(2*np.pi*f0*t_cont)
  plt.plot(t_cont, s_cont, 'lightgray', label='Continuous signal'
     )  # smooth reference curve

  # Sampled signal
  s = np.sin(2*np.pi*f0*t)
  plt.plot(t, s, 'o', color='C0', label='Samples')  # discrete
     samples as dots
```

```
  plt.subplot(1,2,2)
  plt.plot(freqs,np.abs(S),'o')
  plt.show()
```

Example 4

```
img = Image.open("fin.png") #change the path
f = img.convert("L")

# Compute 2D FFT
F = np.fft.fft2(f)

#  Magnitude spectrum (unshifted)
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.imshow(f, cmap='gray')
plt.title('Original image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(np.log(1 + np.abs(F)), cmap='gray')
plt.title('Magnitude spectrum (unshifted)')
plt.axis('off')

#  Shift zero-frequency to center
Fshift = np.fft.fftshift(F)
plt.subplot(2, 2, 3)
plt.imshow(np.log(1 + np.abs(Fshift)), cmap='gray')
plt.title('Magnitude spectrum (shifted)')
plt.axis('off')

#  Inverse FFT (reconstruction)
img_rec = np.fft.ifft2(F)
plt.subplot(2, 2, 4)
plt.imshow(np.real(img_rec), cmap='gray')
plt.title('Reconstructed image')
plt.axis('off')

plt.tight_layout()
plt.show()

# reconstruction error
error = np.abs(f - img_rec)
max_diff = np.max(error)
print(max_diff)
```

Example 5

```
# Parameters
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]

# Generate synthetic "continuous" sinusoidal pattern
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))

#  Define sampling steps (simulate coarser sampling)
sampling_steps = [1, 2, 4, 8, 16]

plt.figure(figsize=(14, 8))
for i, step in enumerate(sampling_steps, 1):
    # 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

    plt.subplot(2, 3, i)
    plt.imshow(img_resized, cmap='gray', vmin=-1, vmax=1)
    plt.title(f'Sampling step = {step}')
    plt.axis('off')

plt.suptitle('Spatial aliasing on a sinusoidal grating', fontsize=14)
plt.tight_layout()
plt.show()
```

Example 6

As the number of samples N increases, the main spectral peak in the DFT becomes narrower. This is because the frequency resolution of the DFT, $\Delta f = f_s/N$, improves with larger N. A longer time window corresponds to finer frequency spacing between bins, so the sinusoidal component is represented by a sharper and taller peak.

For small N, the observation window is short and does not contain an integer number of periods of the sinusoid. The abrupt truncation of the signal acts as multiplication by a rectangular window in the time domain, which corresponds to convolution with a sinc function in the frequency domain. As a result, part of the signal energy spreads into adjacent frequency bins, forming visible side lobes around the main peak.

This spreading of energy away from the true frequency is the visual evidence of spectral leakage. When N increases, the leakage is still present but becomes more localized and

the main lobe becomes thinner.

Example 7 and 8

```
# Parameters
N = 256                  # image size
f_x, f_y = 0.07, 0.12    # spatial frequencies (cycles per pixel)
crop_size = 128          # size of cropped region

# Generate sinusoidal pattern
x = np.arange(N)
y = np.arange(N)
X, Y = np.meshgrid(x, y)
img = np.sin(2 * np.pi * (f_x * X + f_y * Y))

# Compute FFT of full image
F_full = np.fft.fftshift(np.fft.fft2(img))
F_full_mag = np.log(1 + np.abs(F_full))

# Crop (simulate truncation)
img_crop = img[:crop_size, :crop_size]
F_crop = np.fft.fftshift(np.fft.fft2(img_crop))
F_crop_mag = np.log(1 + np.abs(F_crop))

# Apply 2D Hanning window
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))

# Visualization
plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Full sinusoidal image')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(img_crop, cmap='gray')
plt.title('Cropped region (truncation)')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(img_windowed, cmap='gray')
plt.title('Cropped + Hanning window')
plt.axis('off')
```

```
plt.subplot(2, 3, 4)
plt.imshow(F_full_mag, cmap='gray')
plt.title('Spectrum - full image (sharp peaks)')
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(F_crop_mag, cmap='gray')
plt.title('Spectrum - cropped (leakage)')
plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(F_win_mag, cmap='gray')
plt.title('Spectrum - windowed (reduced leakage)')
plt.axis('off')

plt.tight_layout()
plt.show()
```