THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

Bài 1. Khảo sát phổ biên độ–pha & tái tạo ảnh "only phase / only magnitude"

Đề bài

- Cho một ảnh xám bất kỳ (ảnh chụp tài liệu lớp học hoặc ảnh camera lớp), hãy:
- Tính DFT 2D, dịch phổ về tâm (fftshift), hiển thị phổ biên độ dạng log và phổ pha.
- Tái tạo ảnh giữ nguyên **pha** nhưng thay **biên độ** = hằng số; và ngược lại (giữ biên độ, thay pha = 0). So sánh trực quan.

Mục tiêu

- Hiểu ý nghĩa "tần số thấp/cao", vì sao log-magnitude cần thiết để hiển thị.
- Nhìn thấy vai trò "pha" đối với cấu trúc không gian (hình dạng) của ảnh.

Hướng dẫn

- Đọc ảnh, chuẩn hóa kiểu float32.
- fft2 → fftshift → tính magnitude/phase.
- Hiển thị np.log(1+|F|) và phase.
- Ảnh "only phase": đặt magnitude = 1, ghép lại với pha → ifft2.
- Ảnh "only magnitude": đặt phase = 0, giữ magnitude → ifft2.

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

```
import numpy as np, cv2, matplotlib.pyplot as plt
from numpy.fft import fft2, ifft2, fftshift, ifftshift
img = cv2.imread('input.png', cv2.IMREAD_GRAYSCALE).astype(np.float32)/255.0
F = fft2(img)
Fc = fftshift(F)
mag = np.abs(Fc)
pha = np.angle(Fc)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1); plt.title('Log Magnitude'); plt.imshow(np.log1p(mag), cmap='gray'); plt.axis('off')
plt.subplot(1,2,2); plt.title('Phase'); plt.imshow((pha+np.pi)/(2*np.pi), cmap='gray'); plt.axis('off')
plt.show()
# Chỉ pha
Fc_phase_only = np.exp(1j*pha)
img_phase_only = np.real(ifft2(ifftshift(Fc_phase_only)))
# Chỉ biên độ
Fc_mag_only = mag * np.exp(1j*0.0)
img_mag_only = np.real(ifft2(ifftshift(Fc_mag_only)))
def show2(a, b, t1, t2):
    plt.figure(figsize=(10,4))
    plt.subplot(1,2,1); plt.title(t1); plt.imshow(np.clip(a,0,1), cmap='gray'); plt.axis('off')
    plt.subplot(1,2,2); plt.title(t2); plt.imshow(np.clip(b,0,1), cmap='gray'); plt.axis('off')
    plt.show()
show2(img_phase_only, img_mag_only, 'Reconstruction (Only Phase)', 'Reconstruction (Only Magnitude)')
```

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

Bài 2. Khử nhiễu mịn (Gaussian) bằng lọc thông thấp (Ideal/Gaussian/Butterworth)

Đề bài

- Ảnh chụp trong lớp có nhiễu mịn (ISO cao). Hãy khử nhiễu bằng 3 bộ lọc **thông thấp** trong miền tần số: **Ideal LPF**, **Gaussian LPF**, **Butterworth LPF** (n=2,4). Tinh chỉnh tần số cắt và so sánh chất lượng (PSNR/SSIM).

Mục tiêu

- Nắm quy trình: FFT → H(u,v) → nhân phổ → IFFT.
- So sánh biên cắt gắt (Ideal) vs chuyển tiếp mượt (Gaussian/Butterworth).

Yêu cầu

- Thêm scikit-image để đo PSNR/SSIM (khuyến nghị).
- Ảnh gốc (không nhiễu) nếu có để làm “ground truth”; nếu không, so sánh cảm quan + độ sắc nét.

Hướng dẫn

- Viết hàm tạo mặt nạ H(u,v) cho 3 bộ lọc.
- Duyệt nhiều D0 (ví dụ 10, 30, 60 px) và n (2,4) với Butterworth.
- Đo PSNR/SSIM nếu có ground-truth; nếu không thì ước lượng qua trực quan + histogram biên độ.
- Kết luận ngắn: trade-off mờ/giữ chi tiết.

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

```
import numpy as np, cv2, matplotlib.pyplot as plt
from numpy.fft import fft2, ifft2, fftshift, ifftshift
from skimage.metrics import peak_signal_noise_ratio as psnr, structural_similarity as ssim
def make_grids(shape):
    M,N = shape
    u = np.arange(-M//2, M//2)
    v = np.arange(-N//2, N//2)
    V, U = np.meshgrid(v, u)   # chú ý trục
    D = np.sqrt(U**2 + V**2)
    return D

def H_ideal_lp(shape, D0):
    D = make_grids(shape)
    return (D <= D0).astype(np.float32)
def H_gauss_lp(shape, D0):
    D = make_grids(shape)
    return np.exp(-(D**2)/(2*(D0**2)))
def H_butter_lp(shape, D0, n=2):
    D = make_grids(shape)
    return 1.0 / (1.0 + (D/D0)**(2*n))
def freq_filter(img, H):
    F = fftshift(fft2(img))
    G = F * H
    out = np.real(ifft2(ifftshift(G)))
    return np.clip(out, 0, 1)
img = cv2.imread('noisy.png', cv2.IMREAD_GRAYSCALE).astype(np.float32)/255.0
D0 = 30
out_ideal  = freq_filter(img, H_ideal_lp(img.shape, D0))
out_gauss  = freq_filter(img, H_gauss_lp(img.shape, D0))
out_butt2  = freq_filter(img, H_butter_lp(img.shape, D0, n=2))
out_butt4  = freq_filter(img, H_butter_lp(img.shape, D0, n=4))
plt.figure(figsize=(12,6))
titles = ['Original','Ideal LPF','Gaussian LPF','Butter n=2','Butter n=4']
imgs   = [img, out_ideal, out_gauss, out_butt2, out_butt4]
for i,(im,t) in enumerate(zip(imgs,titles),1):
    plt.subplot(2,3,i); plt.title(t); plt.imshow(im, cmap='gray'); plt.axis('off')
plt.tight_layout(); plt.show()
```

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

Bài 3. Khử nhiễu tuần hoàn (sọc/moire) bằng Notch Filter bán tự động

Đề bài

- Một ảnh scan tài liệu có các **vân sọc lặp lại**. Hãy loại bỏ nhiễu tuần hoàn bằng cách:
- Phân tích phổ trung tâm (fftshift) để tìm các **điểm sáng đối xứng** (peaks) do nhiễu.
- Thiết kế **notch reject** (hình "đĩa" loại bỏ quanh peak) với bán kính r nhỏ.
- Áp dụng nhiều notch nếu cần, khôi phục ảnh.

Mục tiêu

- Nhận dạng periodic noise trên phổ và khử bằng notch/band-reject.
- Hiểu khác biệt giữa nhiễu ngẫu nhiên vs tuần hoàn.

Yêu cầu

- Cho phép người dùng click chọn đỉnh nhiễu (hoặc nhập toạ độ), chương trình tạo mặt nạ tự động (đối xứng ±k).

Hướng dẫn

- Tính Fc = fftshift(fft2(img)), hiển thị log-magnitude.
- Chọn toạ độ peak (u0,v0) → sinh mặt nạ notch (0 trong vòng tròn r tại ±(u0,v0), 1 nơi khác).
- Nhân phổ và IFFT.
- Điều chỉnh r/ số notch cho tối ưu (giữ chi tiết chữ, loại sọc).

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

```
import numpy as np, cv2, matplotlib.pyplot as plt
from numpy.fft import fft2, ifft2, fftshift, ifftshift
def notch_mask(shape, centers, r=5):
    M,N = shape
    u = np.arange(-M//2, M//2)
    v = np.arange(-N//2, N//2)
    V, U = np.meshgrid(v, u)
    H = np.ones((M,N), dtype=np.float32)
    for (u0, v0) in centers:
        Dp  = (U-u0)**2 + (V-v0)**2
        Dm  = (U+u0)**2 + (V+v0)**2  # điểm đối xứng
        H[Dp <= r*r] = 0.0
        H[Dm <= r*r] = 0.0
    return H

img = cv2.imread('striped.png', cv2.IMREAD_GRAYSCALE).astype(np.float32)/255.0
F = fftshift(fft2(img))
mag = np.log1p(np.abs(F))
plt.figure(); plt.title('Spectrum (click peaks to read coords)'); plt.imshow(mag, cmap='gray'); plt.axis('off');
plt.show()
# Ví dụ đã xác định được các peak (u0,v0) thủ công
centers = [(30, 45), (-30, 45)]  # thay bằng toạ độ của bạn
H = notch_mask(img.shape, centers, r=7)
G = F * H
out = np.real(ifft2(ifftshift(G)))
plt.figure(figsize=(10,4))
plt.subplot(1,2,1); plt.title('Before'); plt.imshow(img, cmap='gray'); plt.axis('off')
plt.subplot(1,2,2); plt.title('After Notch'); plt.imshow(np.clip(out,0,1), cmap='gray'); plt.axis('off')
plt.show()
```

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

Bài 4. Làm sắc nét ảnh (Unsharp/High-boost vs Laplacian)

Đề bài

- Ảnh bảng viết tay/giáo án bị hơi mờ. Hãy tăng độ nét bằng 2 hướng:
- **Unsharp/High-boost**: tách thành phần mượt (Gaussian LPF) → mask → cộng ngược với hệ số A.
- **Laplacian** trong miền không gian.
- So sánh ảnh sau xử lý và đánh giá viền chữ (không bị "halo" quá mạnh).

Mục tiêu

- Hiểu sharpening là tăng cường **thành phần tần số cao**.
- Biết cách cân A (high-boost) & kernel Laplacian để tránh nhiễu.

Yêu cầu

- Có trackbar hoặc tham số dễ chỉnh: sigma/D0, A (1.2–2.0), kernel Laplacian.

Hướng dẫn

- Unsharp: blur = Gaussian(img), mask = img - blur, sharp = img + A*mask.
- Laplacian: tích chập cv2.filter2D với mặt nạ 4/8-neighbor; cộng trở lại.
- So sánh.

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

```
import numpy as np, cv2, matplotlib.pyplot as plt

img = cv2.imread('blurred.png', cv2.IMREAD_GRAYSCALE).astype(np.float32)/255.0

# Unsharp / High-boost
blur = cv2.GaussianBlur(img, (0,0), sigmaX=1.2)
A = 1.5
unsharp = np.clip(img + A*(img - blur), 0, 1)

# Laplacian sharpening
lap_kernel = np.array([[0,-1,0],[-1,4,-1],[0,-1,0]], dtype=np.float32)
lap = cv2.filter2D(img, -1, lap_kernel)
lap_sharp = np.clip(img + 0.8*lap, 0, 1)

plt.figure(figsize=(12,4))
for i,(im,t) in enumerate([(img,'Original'),(unsharp,'Unsharp/High-boost'),(lap_sharp,'Laplacian')],1):
    plt.subplot(1,3,i); plt.title(t); plt.imshow(im, cmap='gray'); plt.axis('off')
plt.tight_layout(); plt.show()
```

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

Bài 5. Nén ảnh kiểu JPEG nhỏ gọn bằng DCT 8×8 (thực dụng cho tài liệu học liệu)

Đề bài

- Tự cài một bộ nén–giải nén đơn giản kiểu JPEG:
- Chia ảnh xám thành khối 8×8 → DCT 2D từng khối.
- Lượng tử hoá (quantization) với ma trận Q có thể "scale" được.
- Giữ K hệ số đầu (zig-zag) hoặc dùng Q-scale → IDCT để khôi phục.
- Báo cáo tỷ lệ nén (ước tính số hệ số ≠ 0) và SSIM/PSNR.

Mục tiêu

- Thấy khả năng "tập trung năng lượng" của DCT và trade-off nén/chất lượng.

Hướng dẫn

- Chuẩn bị ma trận Q (chuẩn JPEG hoặc tự xây).
- Với mỗi block 8×8: dct → round(dct/Q') → idct (Q' = Q * q_scale).
- Đếm hệ số khác 0 để ước lượng mức nén.
- Đo SSIM/PSNR và hiển thị.

THỰC HÀNH VỀ BIẾN ĐỔI FOURIER

```
import numpy as np, cv2, matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim, peak_signal_noise_ratio as psnr
Q = np.array([
 [16,11,10,16,24,40,51,61],
 [12,12,14,19,26,58,60,55],
 [14,13,16,24,40,57,69,56],
 [14,17,22,29,51,87,80,62],
 [18,22,37,56,68,109,103,77],
 [24,35,55,64,81,104,113,92],
 [49,64,78,87,103,121,120,101],
 [72,92,95,98,112,100,103,99]], dtype=np.float32)
def block_process(img, q_scale=1.0):
    h,w = img.shape
    img = img.astype(np.float32) - 128.0
    out = np.zeros_like(img)
    nonzeros = 0
    for i in range(0,h,8):
        for j in range(0,w,8):
            block = img[i:i+8, j:j+8]
            if block.shape != (8,8): continue
            d = cv2.dct(block)
            q = np.round(d / (Q*q_scale))
            nonzeros += np.count_nonzero(q)
            d_rec = q * (Q*q_scale)
            out[i:i+8, j:j+8] = cv2.idct(d_rec)
    out = np.clip(out + 128.0, 0, 255).astype(np.uint8)
    return out, nonzeros
img = cv2.imread('page.png', cv2.IMREAD_GRAYSCALE)
q_scale = 1.5
rec, nnz = block_process(img, q_scale=q_scale)
ratio = (img.size) / max(nnz,1)  # ước lượng đơn giản
print("Ước lượng tỉ lệ nén ~", ratio)
print("PSNR:", psnr(img, rec, data_range=255))
print("SSIM:", ssim(img, rec, data_range=255))
plt.figure(figsize=(10,4))
plt.subplot(1,2,1); plt.title('Original'); plt.imshow(img, cmap='gray'); plt.axis('off')
plt.subplot(1,2,2); plt.title(f'Reconstructed (q_scale={q_scale})'); plt.imshow(rec, cmap='gray'); plt.axis('off')
plt.show()
```