# Assignment 12

Yehonatan Keypur

August 4, 2024

## 1 Question 1

The complete code for the assignment is attached in a separate file named ‘Q1‘. Please refer to the attached code file for the detailed implementation.

Below is the implementation of the Image Sharpening algorithm:

```python
#laplacian_frequency_filter.py
import...

def load_and_normalize_image(image_path):
    with Image.open(image_path) as img:
        f = np.array(img.convert('L')).astype(float)
    return f / 255

def save_image(image, title, image_path, save_to_dir=False):
    if save_to_dir:
        image_name = image_path.split('/')[-1].split('.')[0]
        output_directory = 'output'
        output_path = f'{output_directory}/{image_name}_{title}.png'
        if image.dtype != np.uint8:
            image = (image * 255).astype(np.uint8)
        Image.fromarray(image).save(output_path)

def display_image(image, title, image_path, save_to_dir=False):
    plt.figure(dpi=150)
    plt.imshow(image, cmap='gray')
    plt.axis('off')
    plt.title(title)
    plt.show()
    save_image(image, title, image_path, save_to_dir)

def fourier_transform(image):
    ft = np.fft.fft2(image)
    return np.fft.fftshift(ft)

def create_laplacian_filter(shape):
    P, Q = shape
    H = np.zeros((P, Q), dtype=np.float32)
    for u in range(P):
        for v in range(Q):
            H[u, v] = -4 * np.pi ** 2 * ((u - P / 2) ** 2 + (v - Q / 2) ** 2)
    return H

def apply_laplacian_filter(ft_image, laplacian_filter):
    Lap = laplacian_filter * ft_image
```

```python
    Lap = np.fft.ifftshift(Lap)
    return np.real(np.fft.ifft2(Lap))

def scale_to_range(image, new_min=-1, new_max=1):
    old_min, old_max = np.min(image), np.max(image)
    old_range = old_max - old_min
    new_range = new_max - new_min
    return (((image - old_min) * new_range) / old_range) + new_min

def enhance_image(original, laplacian, c=-1):
    enhanced = original + c * laplacian
    return np.clip(enhanced, 0, 1)

def laplacian_filter_workflow(image_path, save_to_dir=False):
    f = load_and_normalize_image(image_path)
    display_image(f, "Original_Image", image_path, save_to_dir)
    F = fourier_transform(f)
    ft_magnitude = np.log1p(np.abs(F))
    display_image(ft_magnitude, "Fourier_Transform_Magnitude", image_path, save_to_dir
    )
    H = create_laplacian_filter(F.shape)
    display_image(H, "Laplacian_Filter", image_path, save_to_dir)
    Lap = apply_laplacian_filter(F, H)
    LapScaled = scale_to_range(Lap)
    display_image(LapScaled, "Laplacian_Filtered_Image", image_path, save_to_dir)
    g = enhance_image(f, LapScaled)
    display_image(g, "Enhanced_Image", image_path, save_to_dir)


# main.py
import...

def process_images(image_directory, save_output=True):
    image_dir = Path(image_directory)
    if save_output:
        output_dir = Path('output')
        output_dir.mkdir(exist_ok=True)
    image_extensions = ['.png', '.jpg', '.jpeg', '.tif', '.tiff']
    image_files = [f for f in image_dir.iterdir() if f.suffix.lower() in
    image_extensions]
    for image_file in image_files:
        print(f"Processing {image_file.name}...")
        laplacian_filter_workflow(str(image_file), save_to_dir=save_output)
        print(f"Finished processing {image_file.name}")

if __name__ == "__main__":
    input_directory = Path("/input")
    process_images(input_directory, save_output=True)
```

# 2 Question 2

## 2.1 Two Series Convolution

Given two sequences: $x = \{1, 2, 4, 3\}$ and $h = \{-1, 2, -1\}$, we will calculate $y = h * x$ where $*$ denotes the convolution operation between the sequences.

**Convolution Formula**
The formula for discrete convolution:

$$y[n] = \sum_k h[k] \cdot x[n-k]$$

where the sum is over all $k$ for which both $h[k]$ and $x[n-k]$ are defined.

**Steps**

1. The length of the output sequence $y$ is $N + M - 1$ where $N$ and $M$ are the lengths of $x$ and $h$, respectively.

   $N = 4, \quad M = 3$

   Length of $y = N + M - 1 = 4 + 3 - 1 = 6$

2. Calculate each element of $y$:

$$\begin{aligned}
y[0] &= h[0] \cdot x[0] = (-1) \cdot (1) = -1 \\
y[1] &= h[0] \cdot x[1] + h[1] \cdot x[0] = (-1) \cdot (2) + (2) \cdot (1) = -2 + 2 = 0 \\
y[2] &= h[0] \cdot x[2] + h[1] \cdot x[1] + h[2] \cdot x[0] = (-1) \cdot (4) + (2) \cdot (2) + (-1) \cdot (1) = -4 + 4 - 1 = -1 \\
y[3] &= h[0] \cdot x[3] + h[1] \cdot x[2] + h[2] \cdot x[1] = (-1) \cdot (3) + (2) \cdot (4) + (-1) \cdot (2) = -3 + 8 - 2 = 3 \\
y[4] &= h[1] \cdot x[3] + h[2] \cdot x[2] = (2) \cdot (3) + (-1) \cdot (4) = 6 - 4 = 2 \\
y[5] &= h[2] \cdot x[3] = (-1) \cdot (3) = -3
\end{aligned}$$

**Result**
Therefore, the resulting sequence $y$ is: $y = \{-1, 0, -1, 3, 2, -3\}$

**Final Solution**
The convolution of $x$ and $h$, is $\{-1, 0, -1, 3, 2, -3\}$.

## 2.2 The Fourier Transform of a Gaussian Filter is a Gaussian Function

We will prove that the Fourier transform of a Gaussian filter is a Gaussian function. Specifically, we will show that if:

$$\hat{f}(u,v) = \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)$$

then

$$f(x,y) = 2\pi\sigma^2 \exp\left(-2\pi^2\sigma^2(x^2+y^2)\right).$$

**proof**
We begin with the inverse 2D Fourier transform:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \hat{f}(u,v)e^{2\pi i(ux+vy)}dudv$$

Substituting our given $\hat{f}(u,v)$:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)e^{2\pi i(ux+vy)}dudv$$

This can be separated into two independent integrals:

$$f(x,y) = \int_{-\infty}^{\infty} \exp\left(-\frac{u^2}{2\sigma^2}\right)e^{2\pi iux}du \cdot \int_{-\infty}^{\infty} \exp\left(-\frac{v^2}{2\sigma^2}\right)e^{2\pi ivy}dv$$

Each of these integrals is of the form:

$$I = \int_{-\infty}^{\infty} \exp(-au^2+bu)du$$

where $a = \frac{1}{2\sigma^2}$ and $b = 2\pi ix$ (for the $u$ integral) or $b = 2\pi iy$ (for the $v$ integral).

This integral has a known solution:

$$I = \sqrt{\frac{\pi}{a}}\exp\left(\frac{b^2}{4a}\right)$$

Applying this to our $u$ integral:

$$\int_{-\infty}^{\infty} \exp\left(-\frac{u^2}{2\sigma^2}\right)e^{2\pi iux}du = \sqrt{2\pi\sigma^2}\exp(-2\pi^2\sigma^2x^2)$$

The $v$ integral gives the same result with $y$ instead of $x$. Multiplying these together:

$$\begin{aligned} f(x,y) &= \sqrt{2\pi\sigma^2}\exp(-2\pi^2\sigma^2x^2)\cdot\sqrt{2\pi\sigma^2}\exp(-2\pi^2\sigma^2y^2) \\ &= 2\pi\sigma^2\exp(-2\pi^2\sigma^2(x^2+y^2)) \end{aligned}$$

□

# 3 Question 3

## 3.1 Problem 4.18

### 3.1.1 Part 1: Convolution in time domain ⇔ Multiplication in frequency domain

$$
\begin{aligned}
\mathcal{F}[f(x) \star h(x)] &= \sum_{x=0}^{M-1} \left[ \sum_{m=0}^{M-1} f(m)h(x-m) \right] e^{-j2\pi ux/M} \\
&= \sum_{m=0}^{M-1} f(m) \left[ \sum_{x=0}^{M-1} h(x-m)e^{-j2\pi ux/M} \right] \\
&= \sum_{m=0}^{M-1} f(m)H(u)e^{-j2\pi um/M} \\
&= H(u) \sum_{m=0}^{M-1} f(m)e^{-j2\pi um/M} \\
&= H(u)F(u)
\end{aligned}
$$

### 3.1.2 Part 2: Multiplication in time domain ⇔ Convolution in frequency domain (scaled)

$$
\begin{aligned}
\mathcal{F}[f(x) \cdot h(x)] &= \sum_{x=0}^{M-1} f(x)h(x)e^{-j2\pi ux/M} \\
&= \sum_{x=0}^{M-1} \left[ \frac{1}{M} \sum_{p=0}^{M-1} F(p)e^{j2\pi px/M} \right] \left[ \frac{1}{M} \sum_{q=0}^{M-1} H(q)e^{j2\pi qx/M} \right] e^{-j2\pi ux/M} \\
&= \frac{1}{M^2} \sum_{p=0}^{M-1} \sum_{q=0}^{M-1} F(p)H(q) \sum_{x=0}^{M-1} e^{j2\pi(p+q-u)x/M} \\
&= \frac{1}{M} \sum_{p=0}^{M-1} \sum_{q=0}^{M-1} F(p)H(q)\delta[(p+q-u) \bmod M] \\
&= \frac{1}{M} \sum_{p=0}^{M-1} F(p)H(u-p) \\
&= \frac{1}{M}(F \star H)(u)
\end{aligned}
$$

Where:
$\mathcal{F}$ denotes the Discrete Fourier Transform (DFT),
M is the number of samples,
$\delta[n]$ is the Kronecker delta function,
$\star$ denotes convolution,
F(u) and H(u) are the DFTs of f(x) and h(x) respectively.

## 3.2 Problem 4.20

We will use the sifting property of the 2-D impulse function to prove this statement. Let's begin by recalling the sifting property for a 2-D impulse:

$$\iint f(x,y)\delta(x-x_0, y-y_0)\,dx\,dy = f(x_0, y_0)$$

where $\delta(x-x_0, y-y_0)$ is the 2-D impulse function centered at $(x_0, y_0)$.

Now, let's consider the convolution of a 2-D continuous function $f(t,z)$ with an impulse located at $(t_0, z_0)$:

$$g(t,z) = f(t,z) * \delta(t-t_0, z-z_0)$$

Expanding this using the definition of 2-D convolution:

$$g(t,z) = \iint f(\tau,\zeta)\delta((t-\tau)-t_0, (z-\zeta)-z_0)\,d\tau\,d\zeta$$

We can rewrite this as:

$$g(t,z) = \iint f(\tau,\zeta)\delta((t-t_0)-\tau, (z-z_0)-\zeta)\,d\tau\,d\zeta$$

Now, applying the sifting property of the 2-D impulse (Equation 1):

$$g(t,z) = f((t-t_0), (z-z_0))$$

This result demonstrates that convolving a 2-D continuous function $f(t,z)$ with an impulse $\delta(t-t_0, z-z_0)$ shifts the function so that its origin is moved to the location of the impulse $(t_0, z_0)$.

Specifically:

- For any point $(t,z)$, $g(t,z)$ gives the value of $f$ at the point $(t-t_0, z-z_0)$.
- If $t = t_0$ and $z = z_0$, then $g(t_0, z_0) = f(0,0)$, meaning the origin of $f$ is now at $(t_0, z_0)$.
- If $t_0 = 0$ and $z_0 = 0$ (i.e., the impulse is at the origin), then $g(t,z) = f(t,z)$, meaning the function is not shifted.

## 3.3 Problem 4.24

### 3.3.1 Linearity for 2-D Continuous and Discrete Fourier Transforms

To prove the linearity of 2-D continuous and discrete Fourier transforms, we'll start with the definitions and then show how they satisfy the linearity property.

Definition:

$$F(\mu, \nu) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t, z) e^{-j2\pi(\mu t + \nu z)} \, dt \, dz$$

Linearity property: For the transform to be linear, it must satisfy:

$$F[a_1 f_1(t, z) + a_2 f_2(t, z)] = a_1 F[f_1(t, z)] + a_2 F[f_2(t, z)]$$

Proof:

$$\begin{aligned} F[a_1 f_1(t, z) + a_2 f_2(t, z)] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} [a_1 f_1(t, z) + a_2 f_2(t, z)] e^{-j2\pi(\mu t + \nu z)} \, dt \, dz \\ &= a_1 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_1(t, z) e^{-j2\pi(\mu t + \nu z)} \, dt \, dz \\ &\quad + a_2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_2(t, z) e^{-j2\pi(\mu t + \nu z)} \, dt \, dz \\ &= a_1 F[f_1(t, z)] + a_2 F[f_2(t, z)] \end{aligned}$$

**2-D Continuous Fourier Transform** The last step follows from the distributive property of integrals.

### 3.3.2 2-D Discrete Fourier Transform

Definition:

$$F[u, v] = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f[x, y] e^{-j2\pi(ux/M + vy/N)}$$

Linearity proof:

$$\begin{aligned} F[a_1 f_1[x, y] + a_2 f_2[x, y]] &= \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [a_1 f_1[x, y] + a_2 f_2[x, y]] e^{-j2\pi(ux/M + vy/N)} \\ &= a_1 \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f_1[x, y] e^{-j2\pi(ux/M + vy/N)} \\ &\quad + a_2 \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f_2[x, y] e^{-j2\pi(ux/M + vy/N)} \\ &= a_1 F[f_1[x, y]] + a_2 F[f_2[x, y]] \end{aligned}$$

This uses the distributive property, but for sums instead of integrals.

These proofs demonstrate that both continuous and discrete 2-D Fourier transforms are linear operations, as they preserve addition and scalar multiplication.

The inverse Fourier transforms can be proven linear using the same approach, just working with the inverse transform definitions.

# 4 Question 4

**Theorem**
the convolution of the *sinc* function with itself is equal to the *sinc* function.
That is, $sinc(x) * sinc(x) = sinc(x)$

**proof**
We aim to prove that the convolution of the *sinc* function with itself is equal to the *sinc* function. Let $*$ denote the convolution operation.

Recall the definition of the *sinc* function:

$$sinc(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} & \text{for } x \neq 0 \\ 1 & \text{for } x = 0 \end{cases}$$

The convolution of two functions $f$ and $g$ is defined as:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x - \tau)\, d\tau$$

We will use the Fourier transform method to prove this. Recall that the Fourier transform of $sinc(x)$ is a rectangular function:

$$\mathcal{F}\{sinc(x)\} = rect(f) = \begin{cases} 1 & \text{for } |f| \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

By the convolution theorem, the Fourier transform of a convolution is the product of the Fourier transforms:

$$\mathcal{F}\{sinc(x) * sinc(x)\} = \mathcal{F}\{sinc(x)\} \cdot \mathcal{F}\{sinc(x)\}$$

Applying the Fourier transform of $sinc(x)$:

$$\mathcal{F}\{sinc(x) * sinc(x)\} = rect(f) \cdot rect(f)$$

The product of two identical rectangular functions is the same rectangular function:

$$rect(f) \cdot rect(f) = rect(f)$$

Apply the inverse Fourier transform to both sides:

$$\mathcal{F}^{-1}\{\mathcal{F}\{sinc(x) * sinc(x)\}\} = \mathcal{F}^{-1}\{rect(f)\}$$

The left side simplifies to $sinc(x) * sinc(x)$, and we know that $\mathcal{F}^{-1}\{rect(f)\} = sinc(x)$

Therefore:

$$sinc(x) * sinc(x) = sinc(x)$$

$\square$

—

*Thank you for reviewing this assignment.*