LECTURE 04
FREQUENCY DOMAIN

Punnarai Siricharoen, Ph.D.

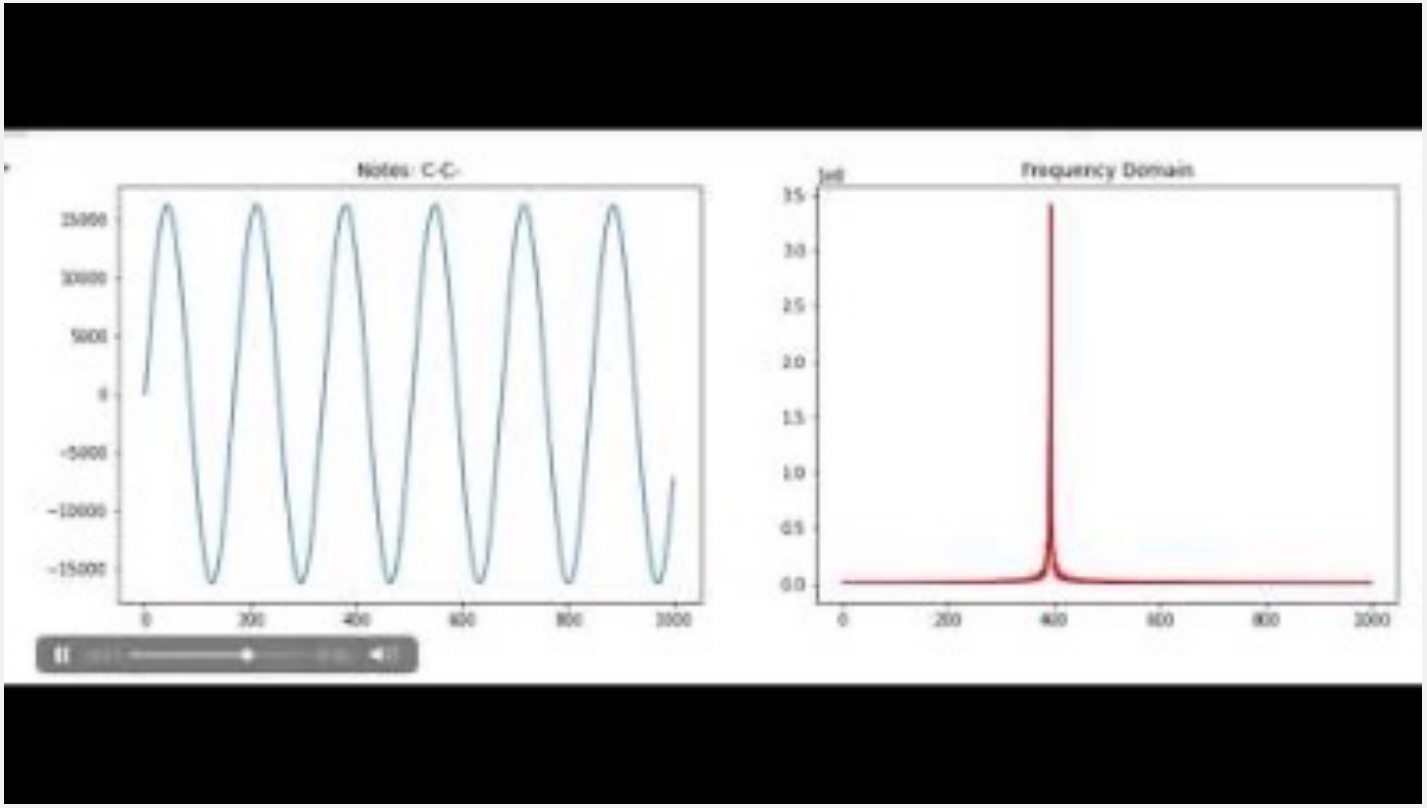
OBJECTIVES

- To understand frequency domain for image analysis
- To be able to describe frequency domain of the image using Fourier transform
- To be able to apply basic filters on an image in frequency domain

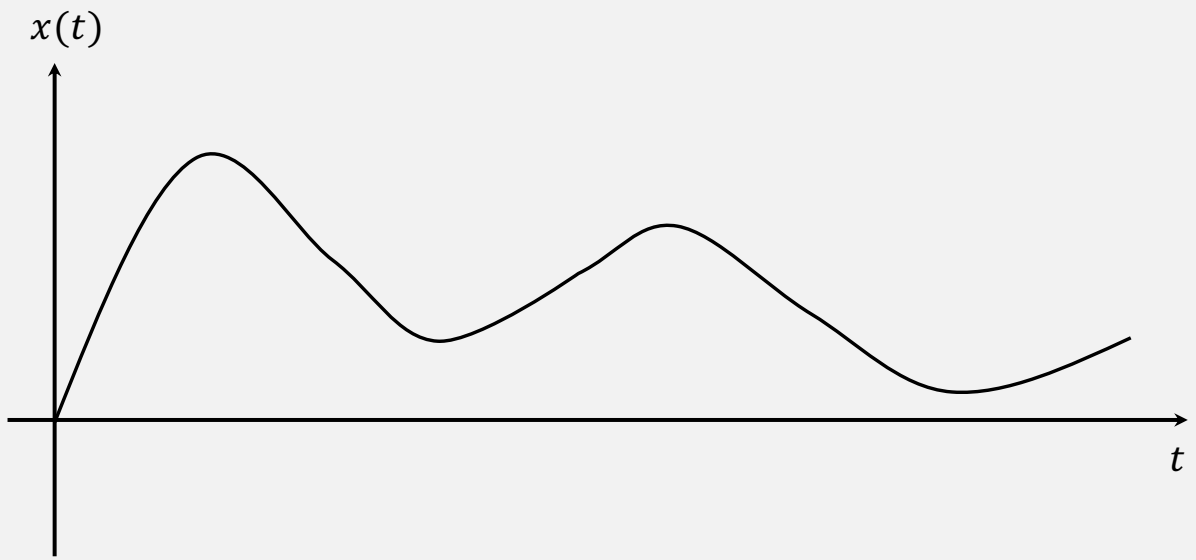
CONTENT

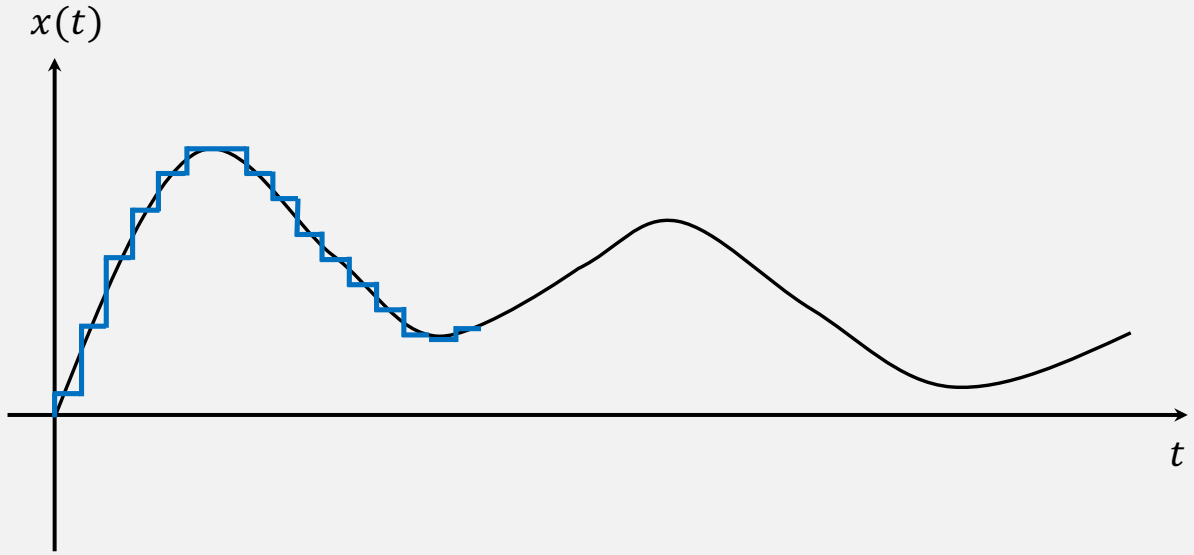
- Fourier Transform (Frequency Domain)
- Filtering in Frequency Domain

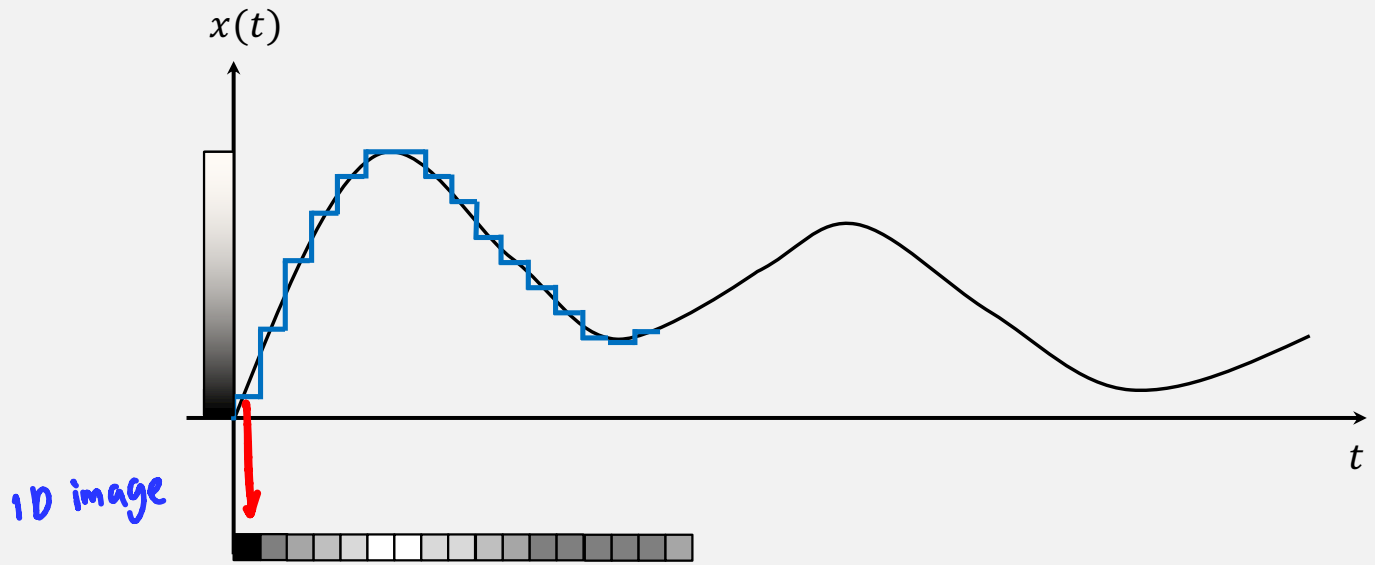
FREQUENCY

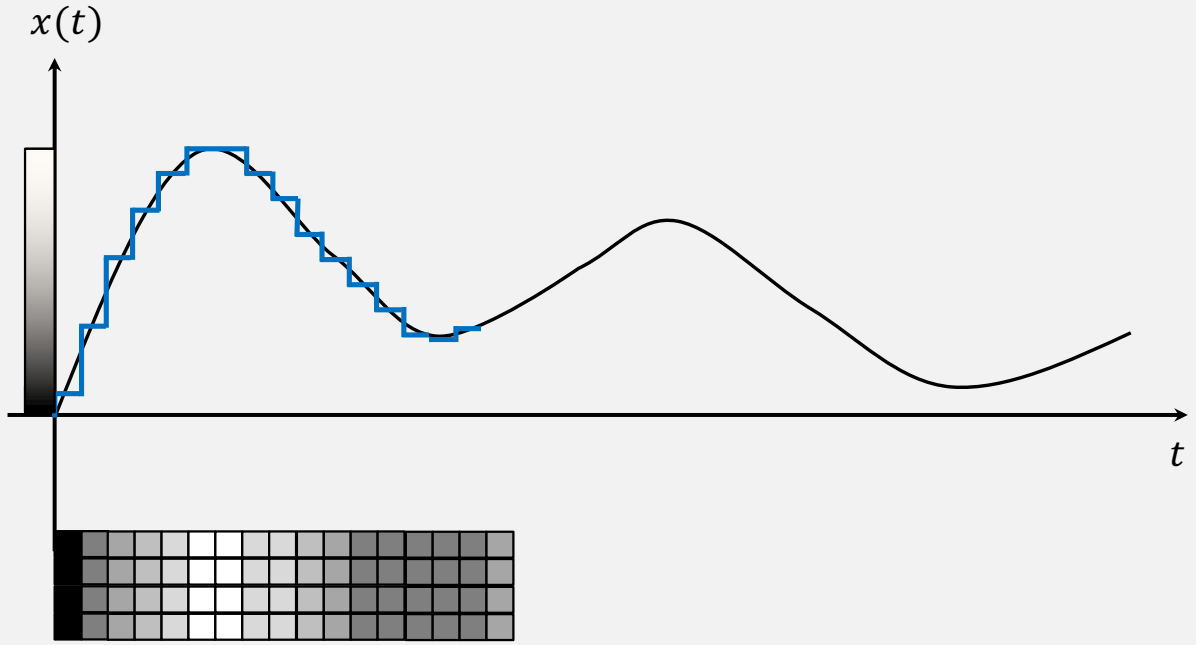

Notes: C-C-
Frequency Domain
1e8
35
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000

10




$x(t)$
t


$x(t)$
t
1D image


$x(t)$
t

FREQUENCY DOMAIN

- **Jean Baptiste Joseph Fourier**'s book, *La Theorie Analitique de la Chaleur (The analytic Theory of Heat), 1807*

- Any function that periodically repeats itself can be expressed as the sum of sines and/or cosines of different frequencies.

FREQUENCY DOMAIN

- It doesn't matter how complicated the function is; as long as it is **periodic** and meets some mathematic conditions. It can be represented by **such as sum**
- Even functions, **not periodic** (area under the curve is finite), can be expressed as integral of **sines/cosines** and **weighted functions**.

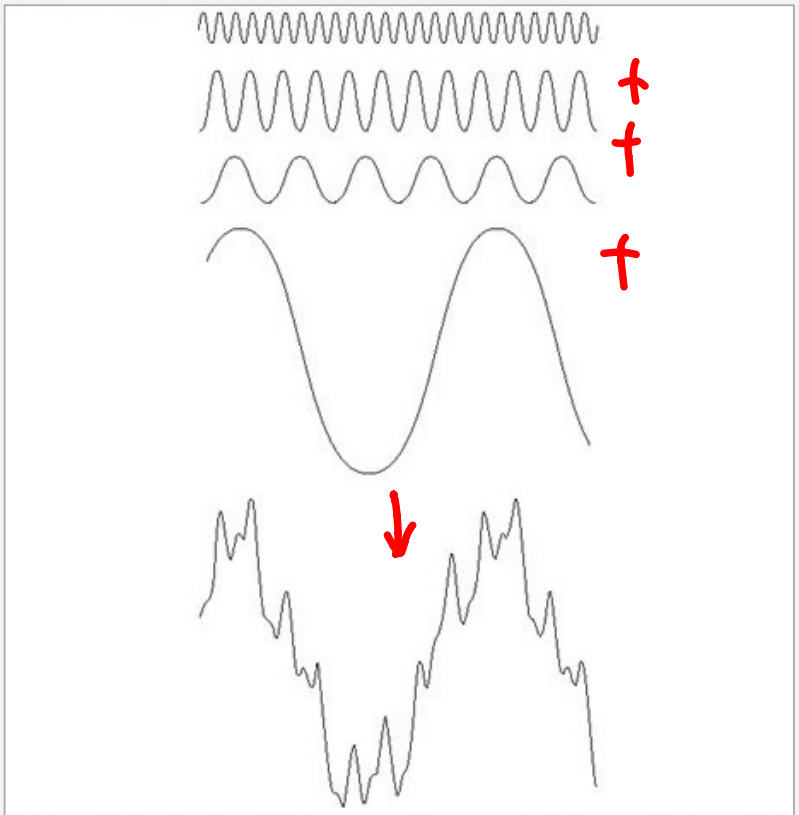
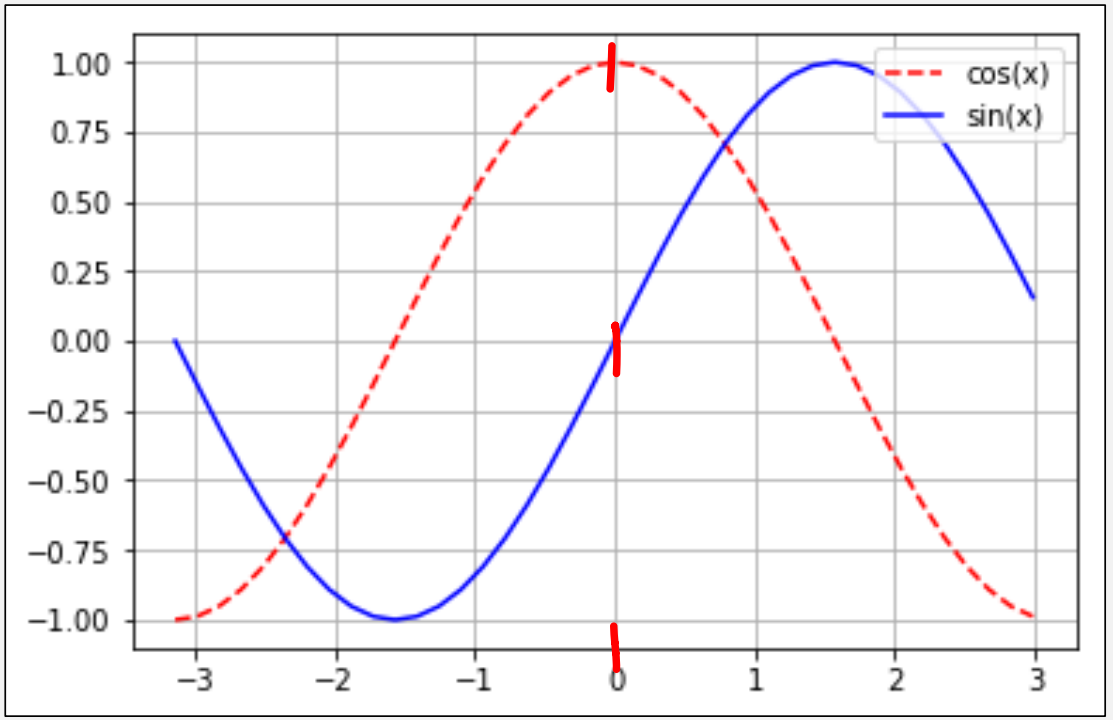


FIGURE 4.1 The function at the bottom is the sum of the four functions above it. Fourier's idea in 1807 that periodic functions could be represented as a weighted sum of sines and cosines was met with skepticism.

FOURIER TRANSFORM

- **Sin/Cos** function

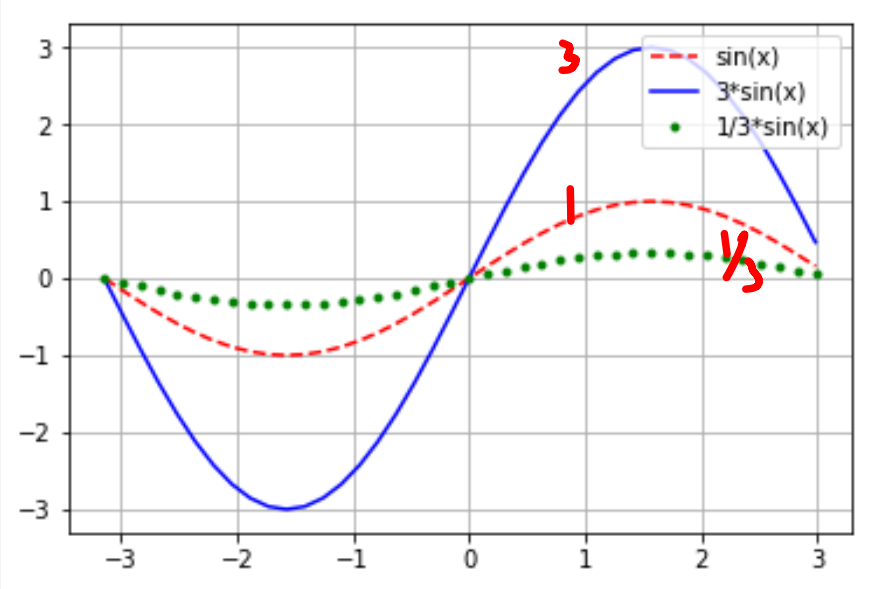
```
import numpy as np
from matplotlib import pyplot as plt

x = np.arange(-np.pi, np.pi, np.pi/20)
f1 = np.cos(x)
f2 = np.sin(x)
plt.plot(x, f1, 'r--')
plt.plot(x, f2, 'b')

plt.legend(('cos(x)', 'sin(x)'),
           loc='upper right')
plt.grid(True)
plt.show()
```



FOURIER TRANSFORM

- **Sin/Cos** function
  - sin(x) vs. $\frac{1}{3}$sin(x) vs. 3*sin(x)

Amp

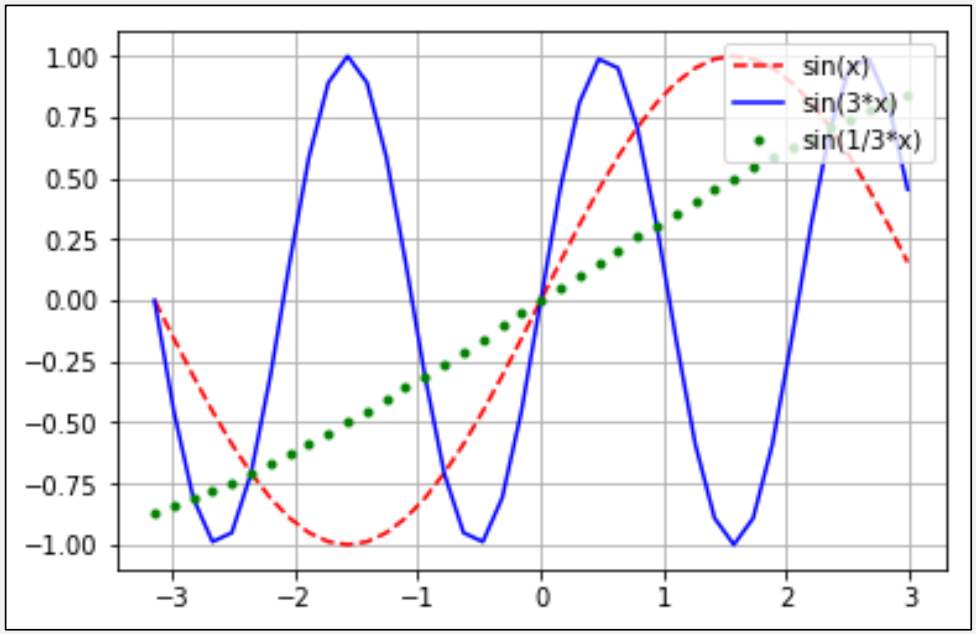


FOURIER TRANSFORM

- **Sin/Cos** function
  - sin(x) vs. sin(x/3) vs. sin(3*x)

time expand

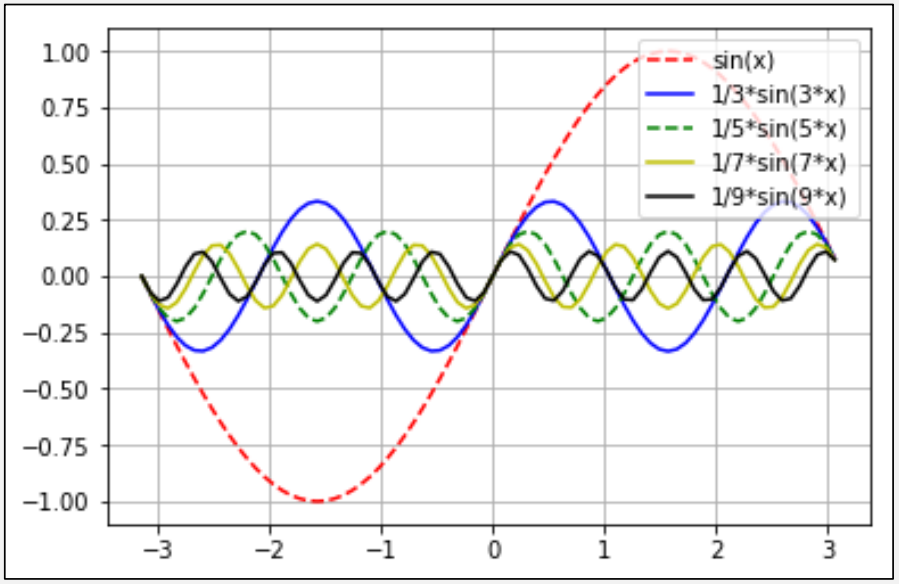
time faster

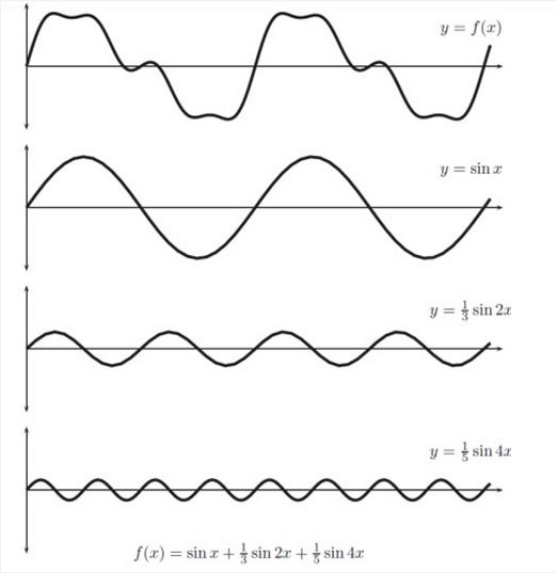


FOURIER TRANSFORM

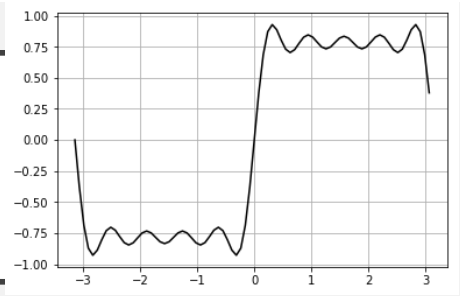
- **A Square wave** has the decomposition:

$$f(x) = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x + \frac{1}{7}\sin 7x + \frac{1}{9}\sin 9x \ \ldots$$



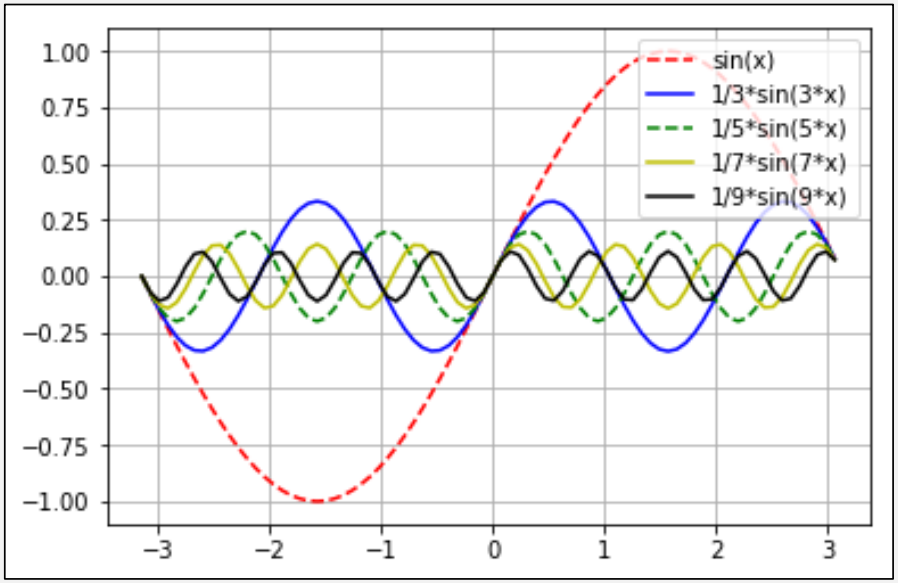


FOURIER TRANSFORM



- **A Square wave** has the decomposition:

$$f(x) = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x + \frac{1}{7}\sin 7x + \frac{1}{9}\sin 9x \ \ldots$$



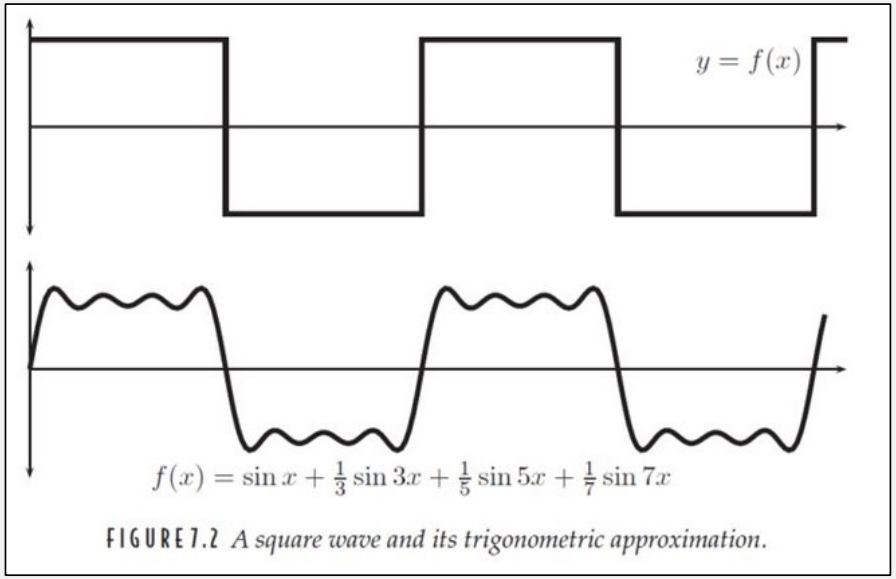
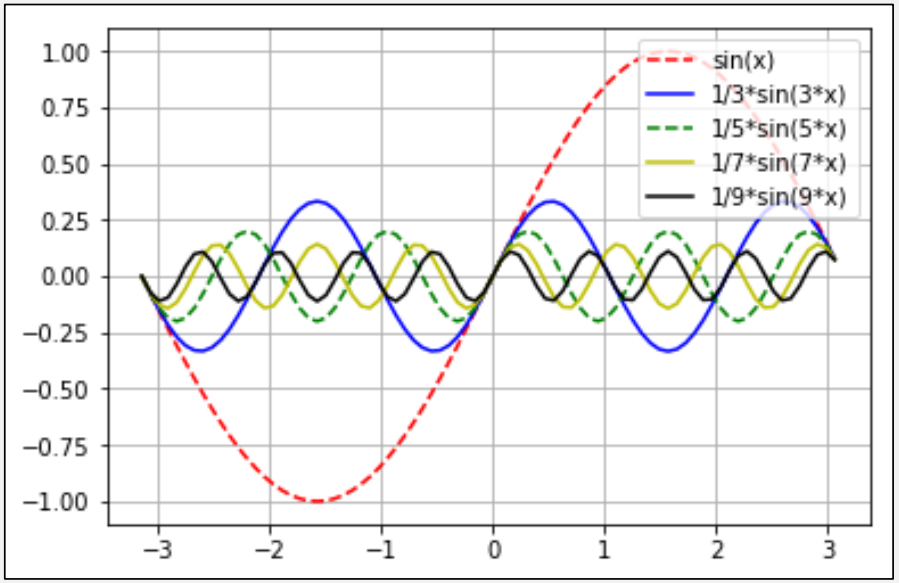


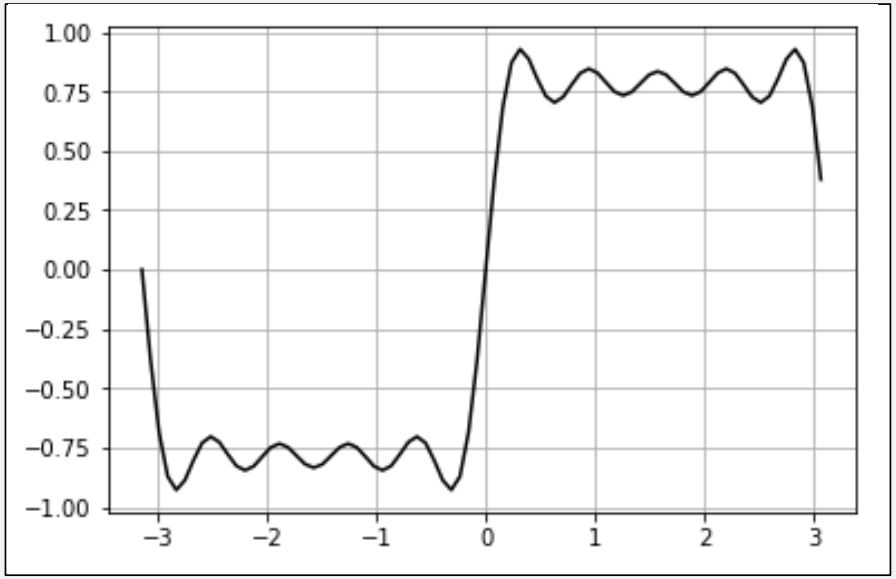
FIGURE 7.2 *A square wave and its trigonometric approximation.*

FOURIER TRANSFORM

- **A Square wave** has the decomposition:

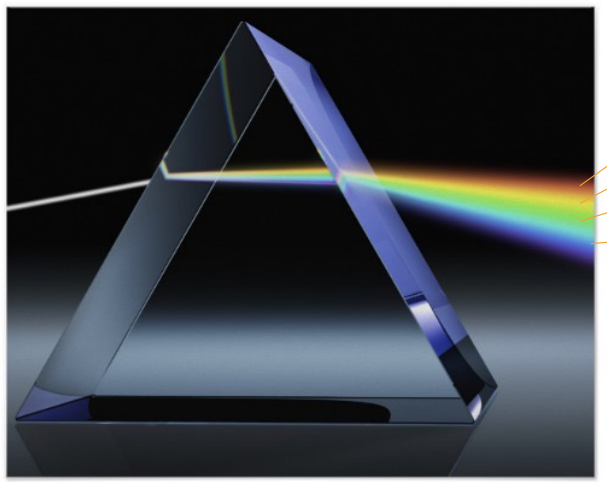
$$f(x) = \sin x + \frac{1}{3}\sin 3x + \frac{1}{5}\sin 5x + \frac{1}{7}\sin 7x + \frac{1}{9}\sin 9x \ \ldots$$

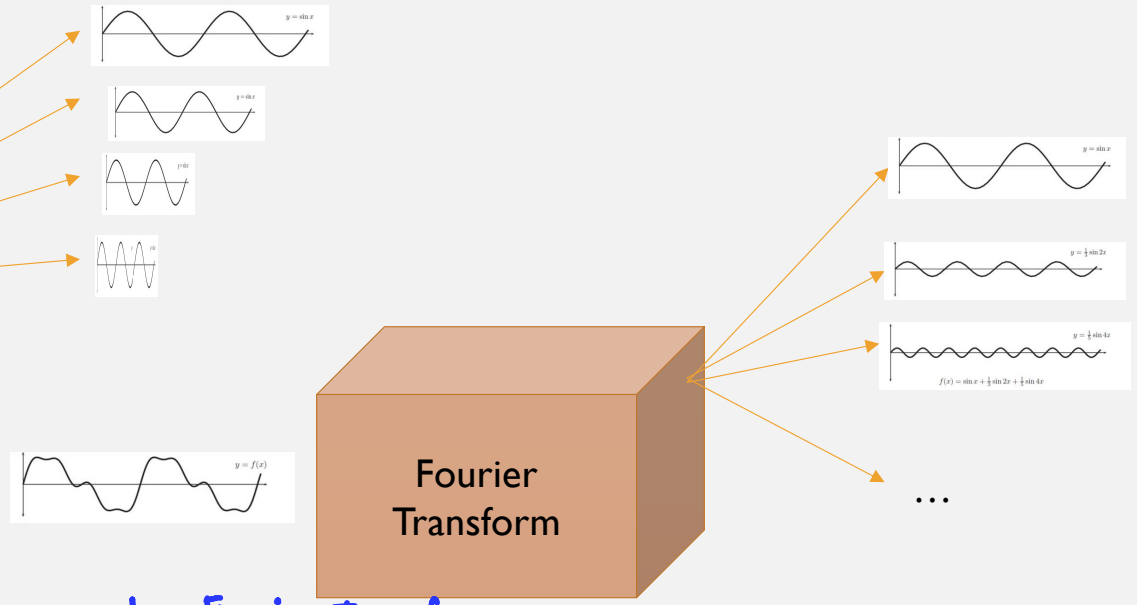




FOURIER TRANSFORM

GLASS PRISM -> separates light into various color components different wavelengths (or frequencies)





using Fourier Transform

MATHEMATICAL PRISM -> decompose a signal into various frequencies.

FOURIER TRANSFORM

- **One-dimensional Fourier Transform :**
- We can obtain **f(x)** by means of the **inverse** Fourier transform,

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du$$

ค่าจริง Amp/coeff freq

- $j = \sqrt{-1}$

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

Euler's formula

FOURIER TRANSFORM

- **One-dimensional Fourier Transform :**
- The Fourier transform, **F(u)** of a single variable defined by the equation

in Amp

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx$$

- $j = \sqrt{-1}$

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)$$

Euler's formula

DISCRETE FOURIER TRANSFORM

- **One-dimensional Fourier Transform :**
- Get original function back using **inverse DFT**:

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M}$$

derive

Compare with the continuous version:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du$$

because image is not cont

Inverse Discrete Fourier transform (Inverse DFT)

for $x = 0,1,2, \ldots, M-1$

- Digital image -> Discrete function of two variables (x,y)

DISCRETE FOURIER TRANSFORM

- **One-dimensional Fourier Transform :**
- Discrete function of one variable, f(x), x = 0,1,2,…,M-1, is given by the equation:

normalize

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-j2\pi ux/M}$$

$e^{j\theta} = \cos\theta + j\sin\theta$

Compare with the continuous version:

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-j2\pi ux}dx$$

Discrete Fourier transform (DFT)

Frequency domain – *u* determines components of the transform.
Each of M terms (0 to M-1) – frequency components

$\cos\omega t / \omega x$

$2\pi f = \omega$

$f = \frac{\omega}{2\pi}$, $f = \frac{2\pi u}{M2\pi} = \frac{u}{M}$

Digital image -> Discrete function of two variables (x,y)

2D DISCRETE FOURIER TRANSFORM

- **Two-dimensional Fourier Transform** of an image f(x,y) of size M x N is given by the equation:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$f_u = \frac{u}{m}$, $f_v = \frac{v}{N}$

- $u = 0,1,2,\ldots,M-1$ — row freq
- $v = 0,1,2,\ldots,N-1$ — col freq

$$\Im[f(x,y)]$$

Digital image -> Discrete function of two variables (x,y)

2D DISCRETE FOURIER TRANSFORM

- **Inverse Fourier Transform** is given by the equation:

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

- $x = 0,1,2,\dots,M-1$
- $y = 0,1,2,\dots,N-1$
- $u \; and \; v$ are frequency variables
- $x \; and \; y$ are spatial (image) variables

Digital image -> Discrete function of two variables (x,y)

2D DISCRETE FOURIER TRANSFORM

- **Fourier Spectrum**: real magnitude of fourier transform

$$|F(u,v)| = [R^2(u,v) + I^2(u,v)]^{1/2}$$

- **Phase angle:**

$$\emptyset(u,v) = \tan^{-1}[\frac{I(u,v)}{R(u,v)}]$$

- **Power spectrum:**

$$\begin{aligned} \mathrm{P}(u,v) &= |F(u,v)|^2 \\ &= R^2(u,v) + I^2(u,v) \end{aligned}$$

- where R(u,v) and I(u,v) are real and imaginary parts of F(u,v)

FOURIER TRANSFORM PROPERTIES

- The value of the transform at (u,v) = (0,0) is

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{0*x}{M}+\frac{0*y}{M})}$$

$$F(0,0) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

No freq

call

Average of grayscale value

DC components of the spectrum

EXERCISE #1 CALCULATE DFT OF A SMALL IMAGE

signal

- [0 200 0 200]

1x4 pixel

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

F

use same size is easier

u = 0
v = 0,1,2,3

$$F(0,0) = \frac{1}{1\times4}(400) = 100$$

$$F(0,1) = \frac{1}{4}\left(0 + 200e^{\frac{-j2\pi(0+\frac{1\cdot1}{4})}{-j}} + 0 + 200e^{\frac{-j2\pi(0+\frac{1\cdot3}{4})}{j}}\right)$$

$\cos(\frac{-\pi}{2}) + j\sin(\frac{-\pi}{2})$

$$= \frac{1}{4}(-200j + 200j) = 0$$

EXERCISE #1 CALCULATE DFT OF A SMALL IMAGE

- [0 200 0 200]

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

$$F(0,2) = \frac{1}{4}\left(0 + 200\, e^{-j2\pi(0+\frac{2}{4})} + 0 + 200\, e^{-j2\pi(0+\frac{6}{4})}\right)$$

$$= \frac{1}{4}\left(200(-1) + 200(-1)\right) = -100$$

$$F(0,3) = \frac{1}{4}\left(0 + 200\, e^{-j\frac{3\pi}{2}} + 0 + 200\, e^{-j2\pi(0+\frac{3\cdot 3}{4})}\right)$$

$$= \frac{1}{4}\left(200j - 200j\right) = 0$$

f(x, y)

$F(u, v)$

low

0	200	0	200
100	0	−100	0

4

$e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$

$w = 2\pi f$

$f_u = \frac{u}{M}$ = 0 เสมอ, เพราะมี row เดียว

$f_v = \frac{v}{M}$

f change every one pixel in an image

$f_{v=1} = \frac{1}{4} = 0.25$

$F_{v=2} = \frac{2}{4} = 0.5$ every 2 pixel in image

pixel/image

$F_{v=3} = \frac{3}{4} = 0.75 = -0.25$

* $F(0,0) = DC$ component *

$F(0,1) = 0$ (No freq)

* $F(0,2) = -100$

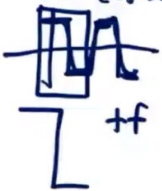
v

neg freq

$F(0,3) = 0$

cos·$w_0 t$

+f



$w_1 = 2\pi \cdot \frac{1}{4} = \frac{2\pi}{4}$

$w_3 = 2\pi \cdot \frac{3}{4} = \frac{6\pi}{4} = \left(2\pi - \frac{2\pi}{4}\right) = -\frac{2\pi}{4}$

No freq

DC		max	
F_0	$F_{v=1}$	$F_{v=2}$	$F_{v=3}$

EXERCISE #1 CALCULATE DFT OF A SMALL IMAGE

- [0 200 0 200 0]

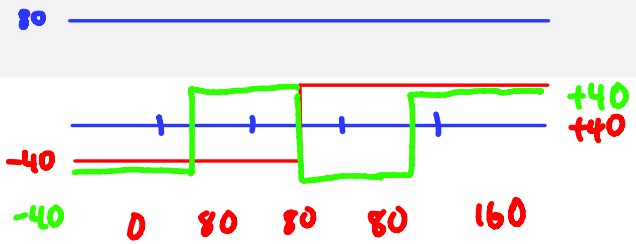
$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

$F(u,v)$ $f_v = 0$ $f_v = \frac{1}{5}$ $f_v = \frac{2}{5}$ $f_v = \frac{3}{5} = \frac{-2}{5}$ $f_v = \frac{4}{5}$

80.00	-20.00 - 14.53j	-20.00 - 61.55j	-20.00 + 61.55j	-20.00 + 14.53j

$F\left(u-\frac{M}{2}, v-\frac{N}{2}\right)$

-20.00 + 61.55j [64.72]	-20.00 + 14.53j [24.72]	80.00	-20.00 - 14.53j [24.72]	-20.00 - 61.55j [64.72]



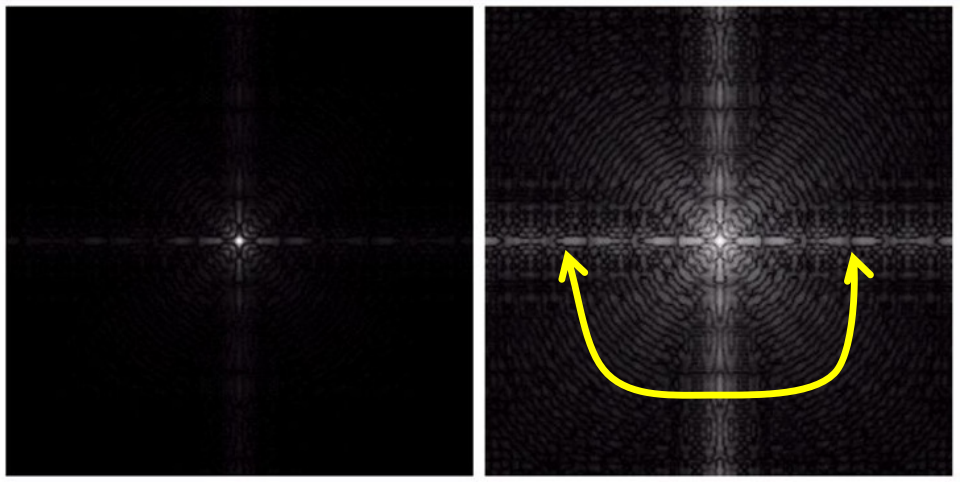
FOURIER TRANSFORM PROPERTIES

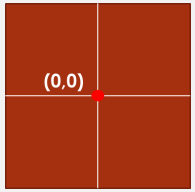
- If f(x,y) is real, its Fourier transform is conjugate symmetric:

$$F(u,v) = F^*(-u,-v)$$

a b

FIGURE 3.5
(a) Fourier spectrum.
(b) Result of applying the log transformation given in Eq. (3.2-2) with $c = 1$.





$$|F(u,v)| = |F(-u,-v)|$$

FOURIER TRANSFORM PROPERTIES

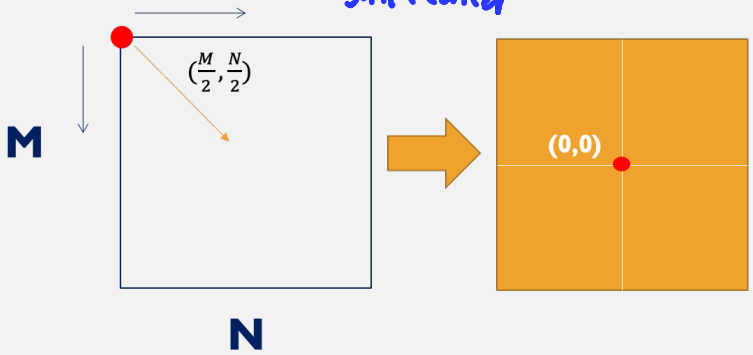
- Shifting:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

original Image → F(u,v) → shift center

or $(-1)^{x+y}$ · Image → F(u,v) → shift center

$$F\left(u-\frac{M}{2}, v-\frac{N}{2}\right) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{(u-\frac{M}{2})x}{M}+\frac{(v-\frac{N}{2})y}{N}\right)}$$



$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{(u-\frac{M}{2})x}{M}+\frac{(v-\frac{N}{2})y}{N}\right)}$$

$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\, e^{j\pi(x+y)}\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$\cos(\pi) + j\sin(\pi)$

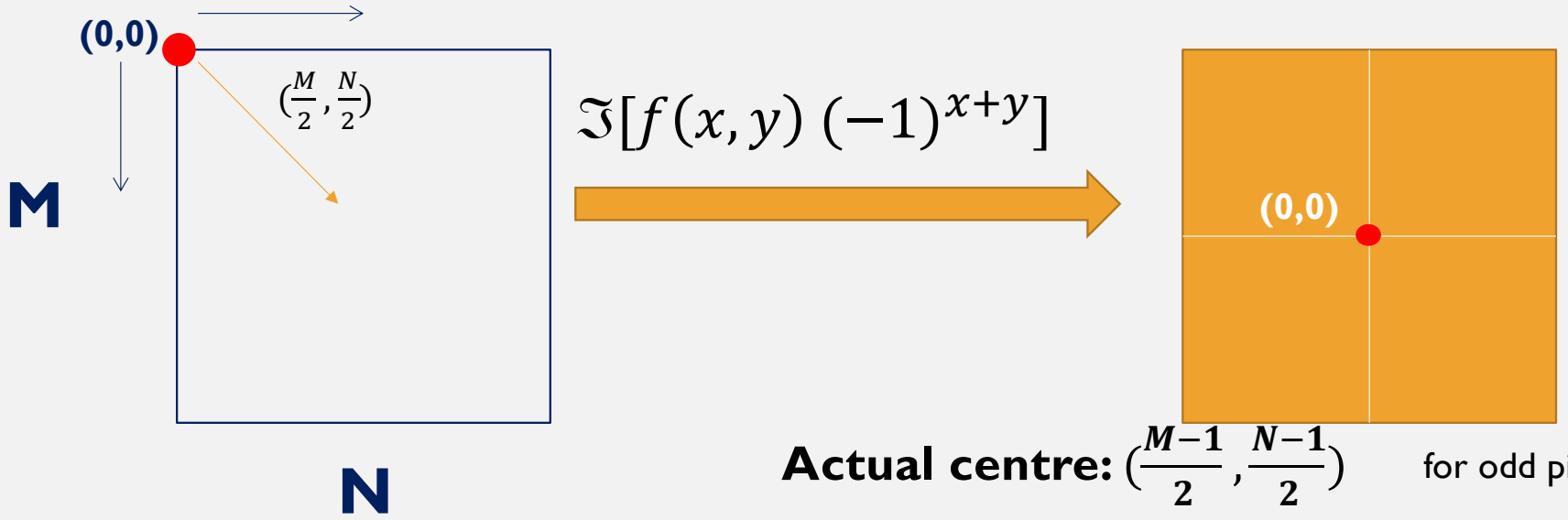
$$= \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)(-1)^{(x+y)}\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

$$F(u',v') = F\left(u-\frac{M}{2}, v-\frac{N}{2}\right)$$

where $F(u',v')$ is a shifted version of $F(u,v)$

FOURIER TRANSFORM PROPERTIES

- Multiply with $(-1)^{x+y}$ to $f(x,y)$ to shift the origin of FT $F(u,v)$ from $(0,0)$ to $(\frac{M}{2},\frac{N}{2})$



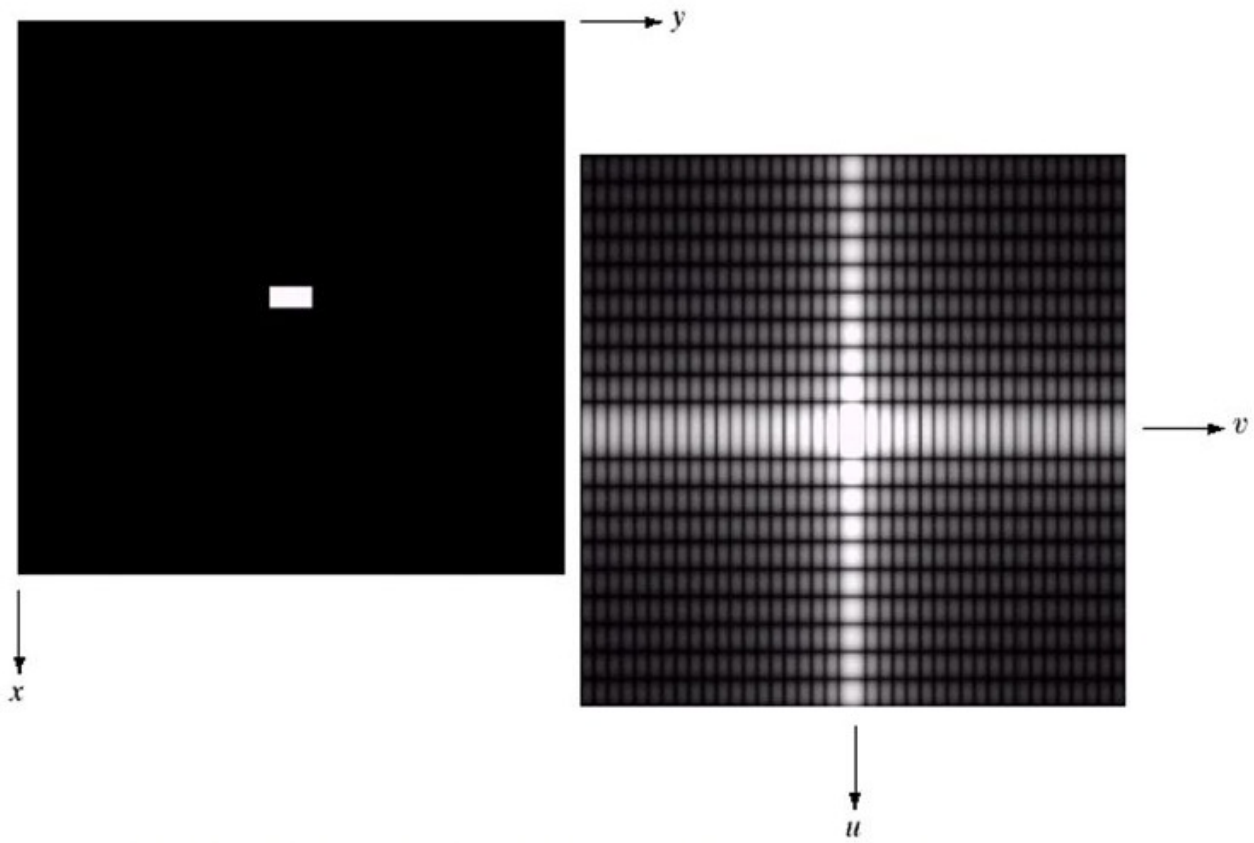
Actual centre: $(\frac{M-1}{2},\frac{N-1}{2})$ for odd pixel size

$(\frac{M}{2},\frac{N}{2})$ for even pixel size

FOURIER TRANSFORM PROPERTIES

a b

FIGURE 4.3
(a) Image of a 20×40 white rectangle on a black background of size 512×512 pixels.
(b) Centered Fourier spectrum shown after application of the log transformation given in Eq. (3.2-2). Compare with Fig. 4.2.



FOURIER TRANSFORM

- The advent of **fast Fourier transform (FFT)** algorithms in the late 1950s.
  - Practical processing & meaningful interpretation.

- Images: functions of finite duration
- Fourier transform – feature extraction, image enhancement approaches.

PYTHON: COMPUTE FOURIER TRANSFORM

- fft – Fast Fourier Transform (DFT) of a vector
- ifft – inverse DFT of a vector
- fft2 – DFT of a matrix
- ifft2 – inverse DFT of a matrix
- fftshift – shift DFT origin to image centre

PYTHON: COMPUTE FOURIER TRANSFORM

High or Low frequency?

- E.g., >> image_simple = np.ones((8,8),dtype=np.uint8)

```
image_simple
Out[47]:
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]], dtype=uint8)
```

```
f.astype(np.uint8)
Out[51]:
```

DC coefficient is sum of all the matrix value

```
array([[64,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0]], dtype=uint8)
```

PYTHON: COMPUTE FOURIER TRANSFORM

High or Low frequency?

- E.g., >> image_simple = np.ones((8,8),dtype=np.uint8)

image_simple

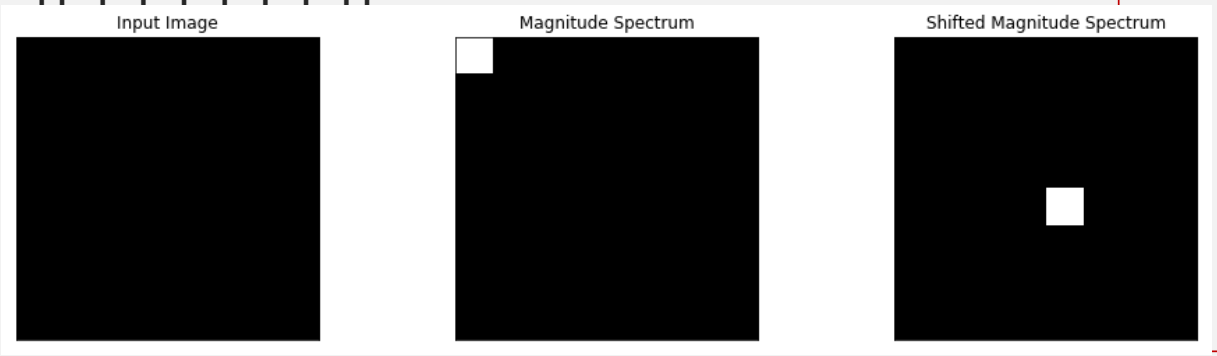
Out[47]:

array([[1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1]

```
f = np.fft.fft2(image_simple)
fshift = np.fft.fftshift(f)
fshift.astype(np.uint8)
```

DC coefficient is sum of all the matrix value

```
array([[ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 64,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0]], dtype=
```

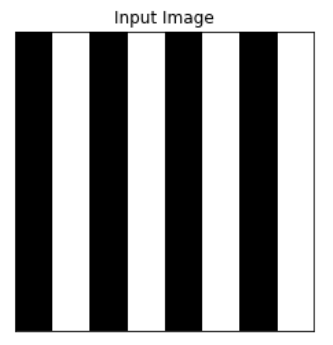


PYTHON: COMPUTE FOURIER TRANSFORM

High or Low frequency?

E.g.,

```
image_simple2 = np.array([(0,200), (0,200)])
image_simple2 = np.tile(image_simple2,(4,4))
array([[  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200]])
```



- f = np.fft.fft2(image_simple2)
- fshift = np.fft.fftshift(f)
- magnitude_spectrum = np.log(1+np.abs(fshift))

```
array([[ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ],
       [ 8.76420951,  0.       ,  0.       ,  0.       ,  8.76420951,
         0.       ,  0.       ,  0.       ],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ]])
```

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

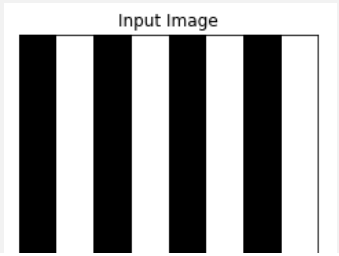
FFT LIBRARY omits term "1/MN"

PYTHON: COMPUTE FOURIER TRANSFORM

High or Low frequency?

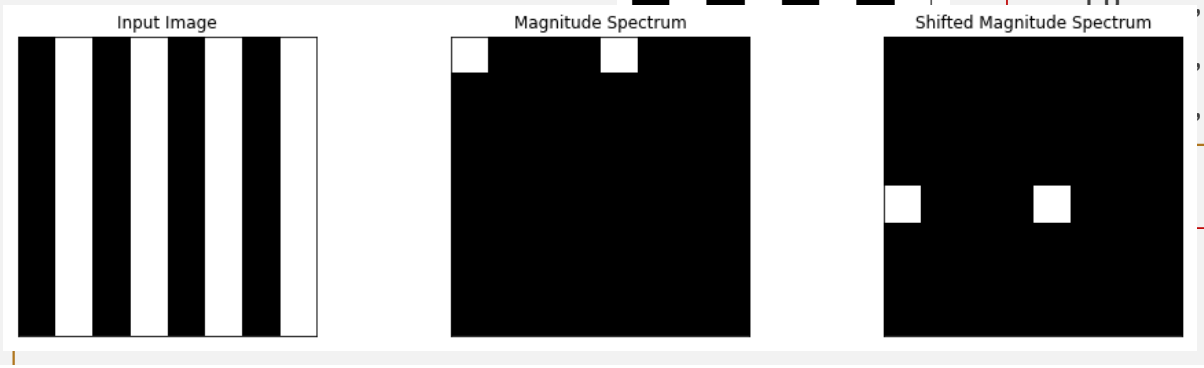
E.g.,

```
image_simple2 = np.array([(0,200), (0,200)])
image_simple2 = np.tile(image_simple2,(4,4))
array([[  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
       [  0, 200,   0, 200,   0, 200,   0, 200],
```



- f = np.fft.fft2(image_simple2)
- fshift = np.fft.fftshift(f)
- magnitude_spectrum = np.log(1+np.abs(fshift))

```
array([[ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
       [ 8.76420951,  0.       ,  0.       ,  0.       ,  8.76420951,
         0.       ,  0.
],
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
       [ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
         0.       ,  0.       ,  0.       ,  0.
```



FFT LIBRARY omits term "1/MN"

EXERCISE #2.1 COMPUTE DFT USING FFT LIBRARY

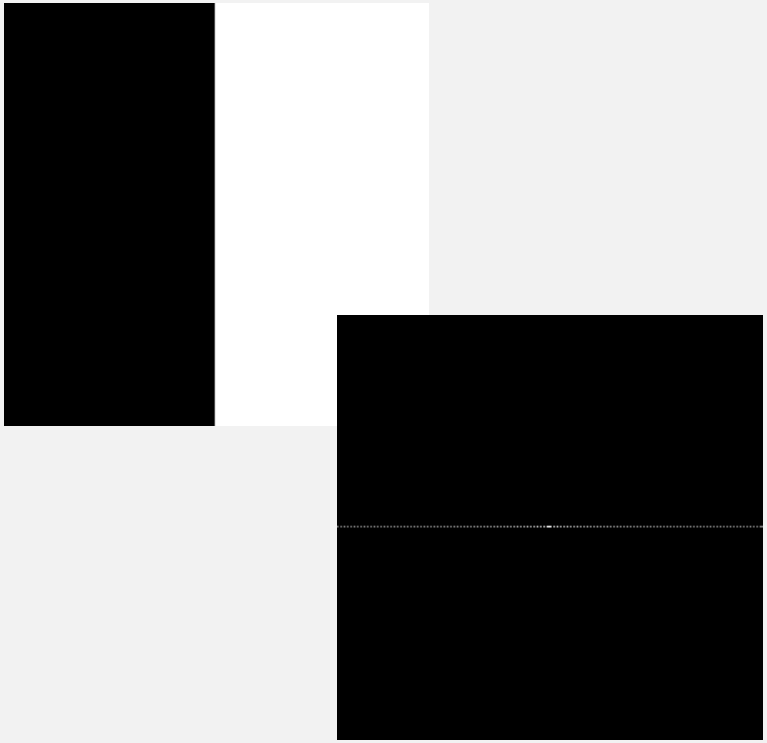
- Compute Fourier transform and shift Fourier transform of

```
a = np.zeros((256,128),dtype=np.uint8)
b = np.ones((256,128),dtype=np.uint8)
image_simple = np.concatenate((a,b), axis = 1)
```

Compare the results with the DFT of **a** (previous example)

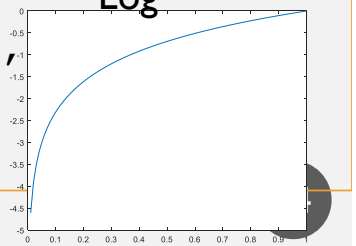
PYTHON: COMPUTE DFT OF A BLACK-AND-WHITE IMAGE

- DFT of images



```
a = np.zeros((256,128),dtype=np.uint8)
b = np.ones((256,128),dtype=np.uint8)
image_simple = np.concatenate((a,b), axis = 1)
f = np.fft.fft2(image_simple)
fshift = np.fft.fftshift(f)
magnitude_spectrum = np.log(1+np.abs(fshift))

fig = plt.figure(figsize=(16,4))
plt.subplot(121),plt.imshow(image_simple.astype(np.u
int8), cmap = 'gray')
plt.title('Input Image'), plt.xticks([]),
plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum.astyp
e(np.uint8), cmap = 'gray')
plt.title('Shifted Magnitude Spectrum'),
plt.xticks([]), plt.yticks([])
plt.show()
```

Log



EXERCISE #2.2 FIND DFT ON A BOX IMAGE

- Create a box image, compute and display DFT of the image

original image

```
image_simple2 = np.zeros((256,256),dtype=np.uint8)
image_simple2[78:179,78:179] = 1
```

numpy.linspace(*start, stop, num=50, endpoint=True, retstep=False, dtype=None*)¶

Return evenly spaced numbers over a specified interval.

EXERCISE #2.3 FIND DFT ON ANOTHER BOX IMAGE

- Create another box image, compute and display DFT of the image

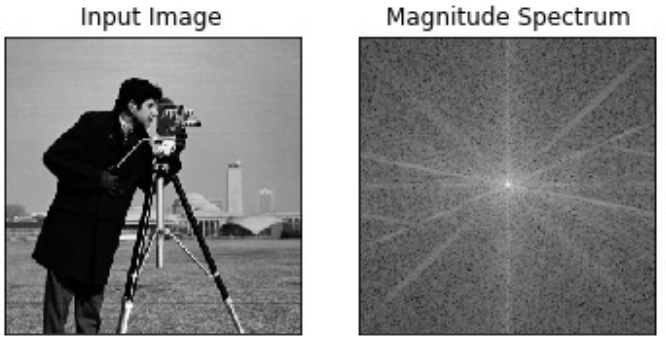
original image

```
nx, ny = (256, 256)
x = np.linspace(0, nx-1, nx)
y = np.linspace(0, ny-1, ny)
i, j = np.meshgrid(x, y, sparse=True)
rotate_box = np.zeros((256,256),dtype=np.bool)
rotate_box = (i+j<329)&(i+j>182)&(i-j>-67)&(i-j<73)



image_simple3 = rotate_box.astype(np.uint8)
```

meshgrid() is very useful to evaluate functions on a grid.

DFT ON ANOTHER IMAGES

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('cameraman.tif',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = np.log(1+np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



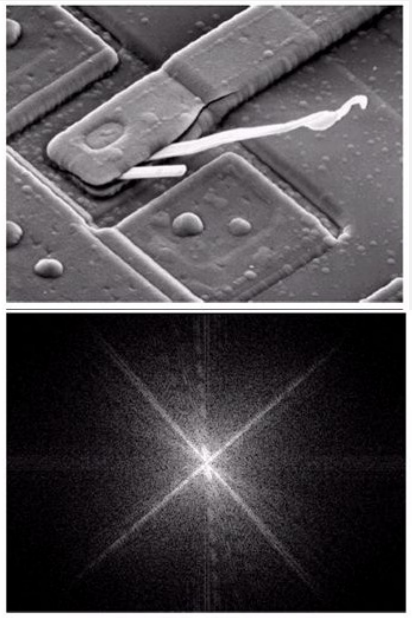
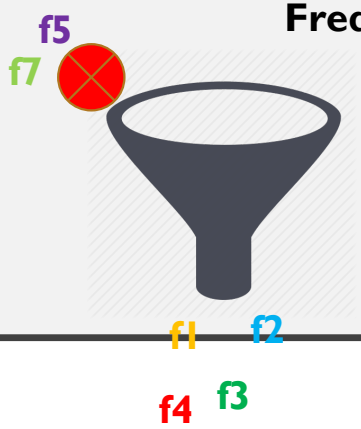


FIGURE 4.4
(a) SEM image of a damaged integrated circuit. (b) Fourier spectrum of (a). (Original image courtesy of Dr. J. M. Hudak, Brockhouse Institute for Materials Research, McMaster University, Hamilton, Ontario, Canada.)



FILTERING IN FREQUENCY DOMAIN

APPLICATIONS: FREQUENCY DOMAIN FILTERING

- Ideal filtering
- Butterworth filtering
- Gaussian filtering

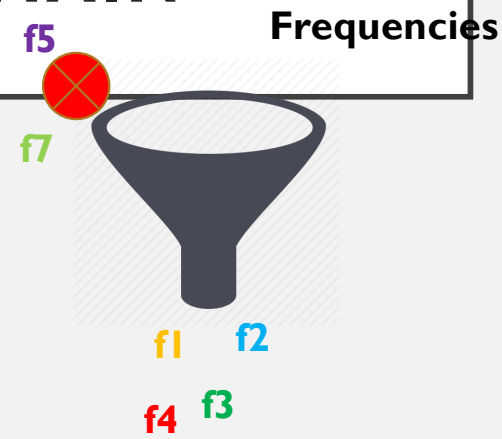
Applications:

- 1) Smoothing Frequency-Domain Filters

  -> low pass filters
- 2) Sharpening Frequency-Domain Filters

  -> high pass filters

FILTERING IN THE FREQUENCY DOMAIN



- ***Filter***: suppresses certain frequencies while leaving others unchanged.

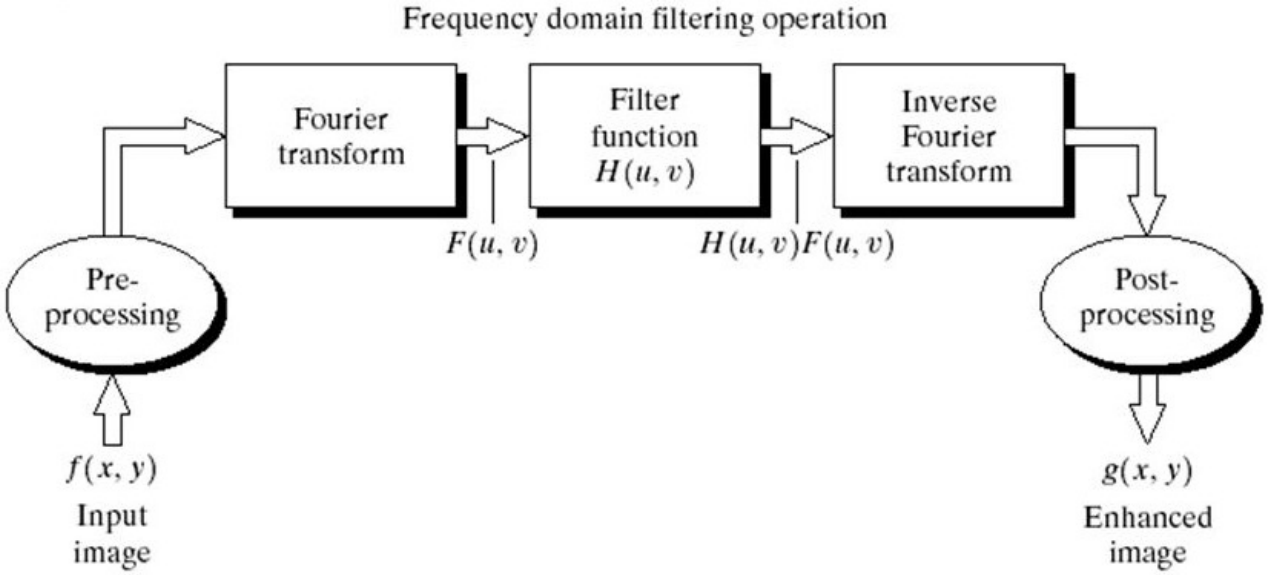


FIGURE 4.5 Basic steps for filtering in the frequency domain.

1) Multiply the input image by $(-1)^{x+y}$
2) Compute $F(u, v)$ (DFT)
3) Multiply $F(u, v)$ by a filter $H(u, v)$
4) Compute iDFT
5) Obtain the real part
6) Multiply the result by $(-1)^{x+y}$

FILTERING IN THE FREQUENCY DOMAIN

- Basic filtering model:

เหตุผลที่ทำ เพราะง่ายกว่าใน spatial domain

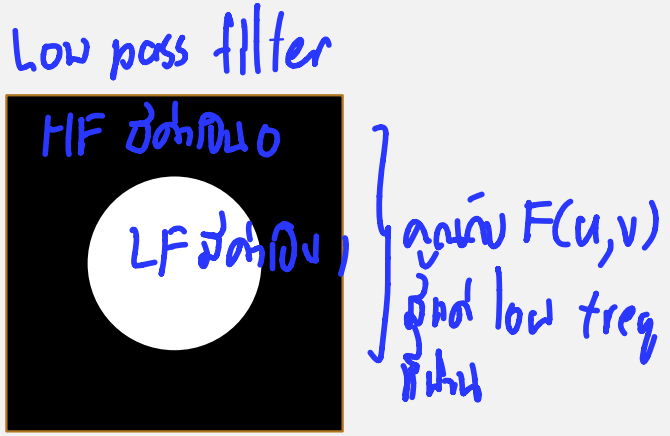
cheaper, easier

$$G(u,v) = H(u,v)F(u,v)$$

- where F(u,v) is FT of the image. H(u,v) – a filter transfer function

LOW-PASS FILTERS (LPF)

- LPF – attenuate the high frequency components -> blurred images -> **edges** and other **abrupt changes** in gray levels are associated with the **high-frequency components.**

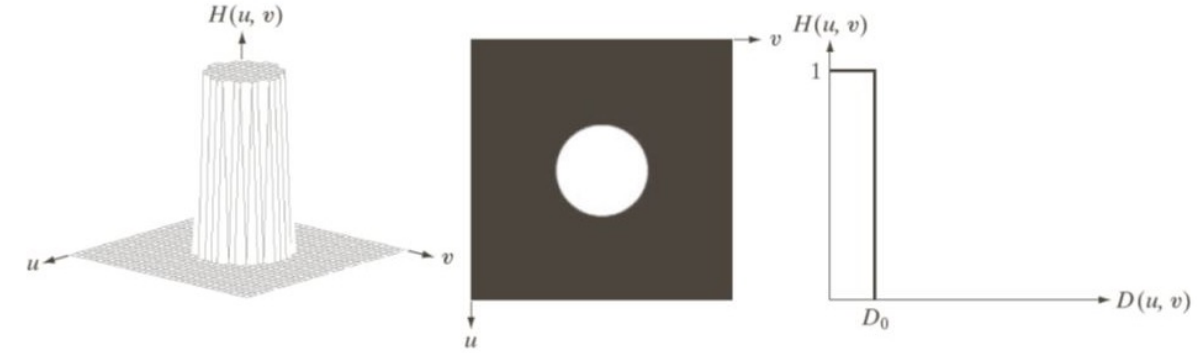


IDEAL LOW-PASS FILTERING

- Ideal low-pass matrix is a binary matrix defined by

$$H(u,v) = \begin{cases} 1 & if\ D(u,v) \leq D_0 \\ 0 & if\ D(u,v) > D_0 \end{cases}$$

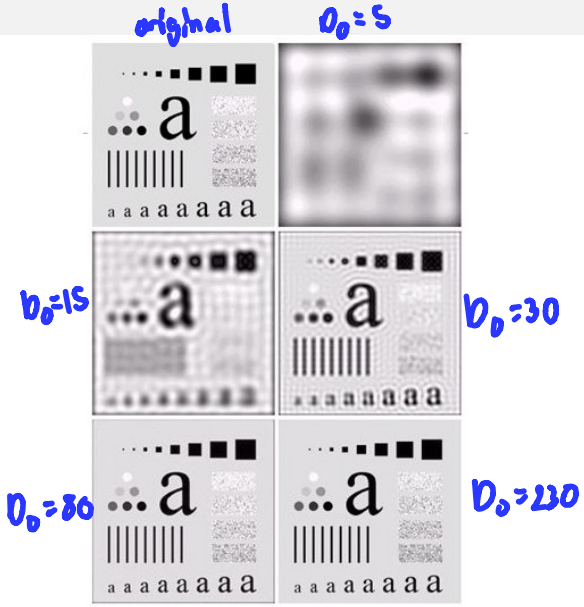
D_0 : Cutoff Frequency
$D(u,v)$: Distance from Fourier origin (assume to be **shifted**)



a b c

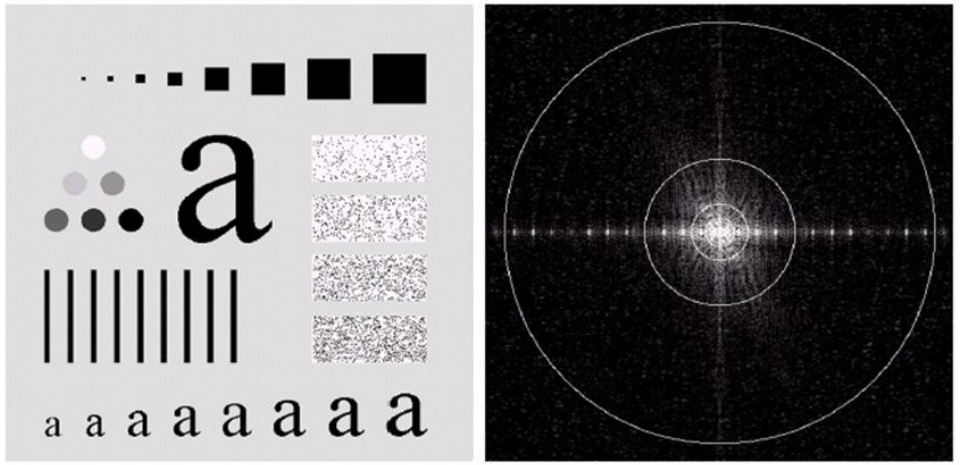
FIGURE 4.40 (a) Perspective plot of an ideal lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross section.

IDEAL LOW-PASS FILTERING



a b
c d
e f

FIGURE 4.12 (a) Original image. (b)–(f) Results of ideal lowpass filtering with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). The power removed by these filters was 8, 5.4, 3.6, 2, and 0.5% of the total, respectively.



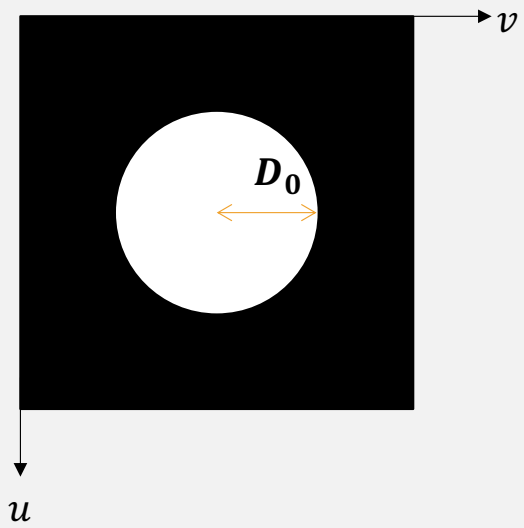
a b

(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2nd Edition.

FIGURE 4.11 (a) An image of size 500 × 500 pixels and (b) its Fourier spectrum. The superimposed circles have radii values of 5, 15, 30, 80, and 230, which enclose 92.0, 94.6, 96.4, 98.0, and 99.5% of the image power, respectively.

BUTTERWORTH LOW-PASS FILTERS

- With D_0 cutoff frequency, transfer function of a Butterworth low-pass filter (BLPF) is defined as

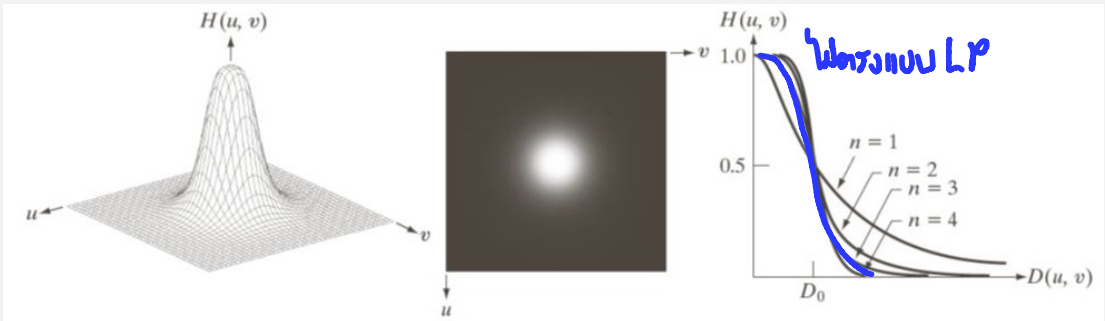


$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}$$

degree

distance

n : order

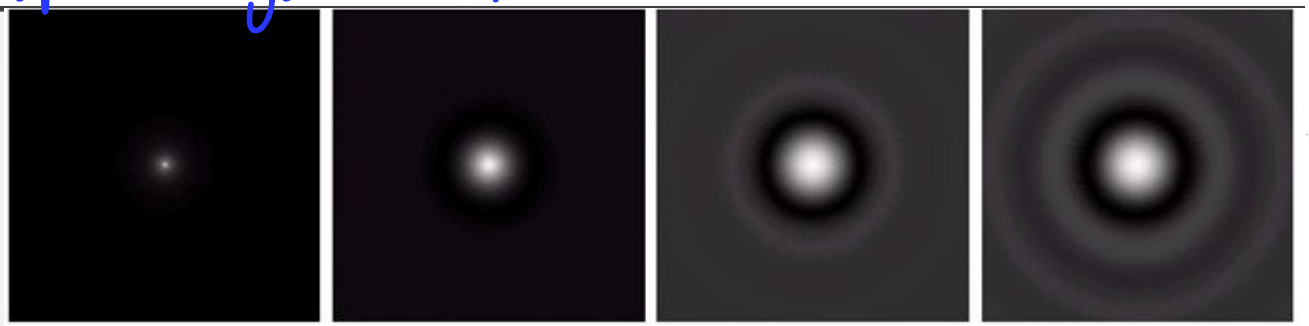


a b c

FIGURE 4.44 (a) Perspective plot of a Butterworth lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

BUTTERWORTH LOW-PASS FILTERS

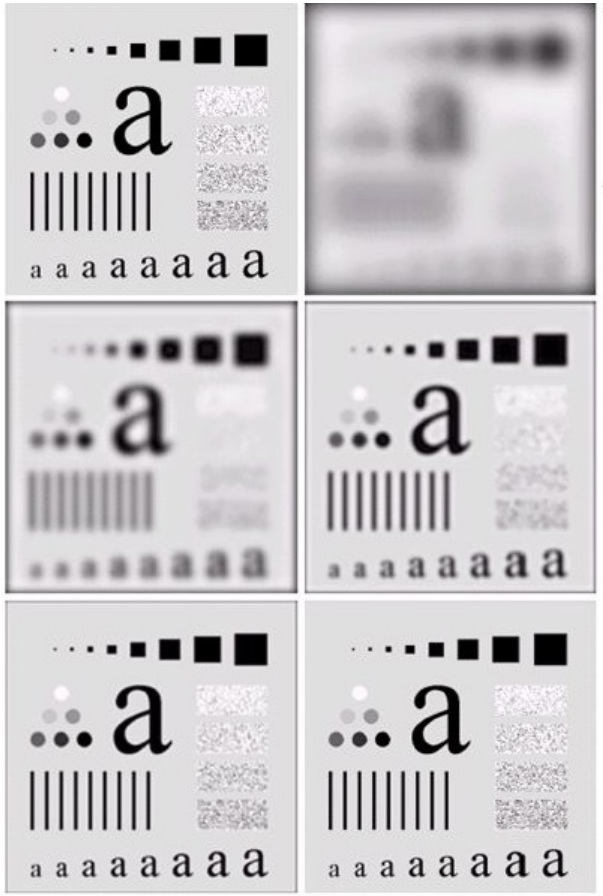
Spatial Image of BLPF



sinc func

a b c d

FIGURE 4.16 (a)–(d) Spatial representation of BLPFs of order 1, 2, 5, and 20, and corresponding gray-level profiles through the center of the filters (all filters have a cutoff frequency of 5). Note that ringing increases as a function of filter order.



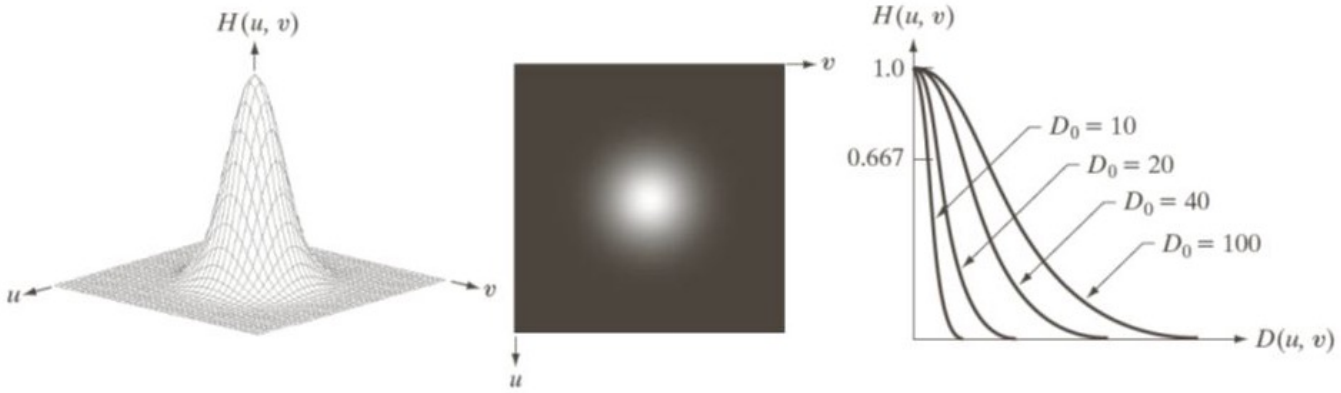
a b c d e f

FIGURE 4.15 (a) Original image. (b)–(f) Results of filtering with BLPFs of order 2, with cutoff frequencies at radii of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Fig. 4.12.

GAUSSIAN LOW-PASS FILTERS

- Transfer function:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$



a b c

FIGURE 4.47 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.

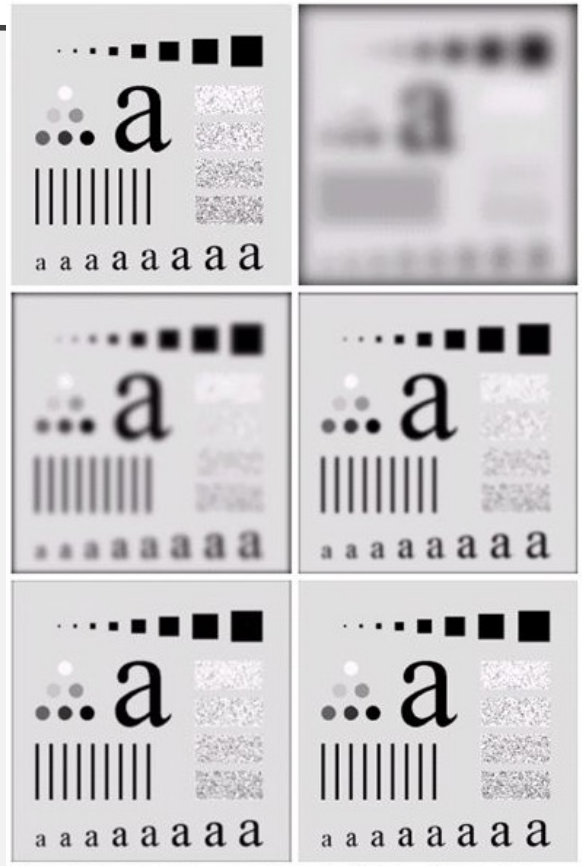
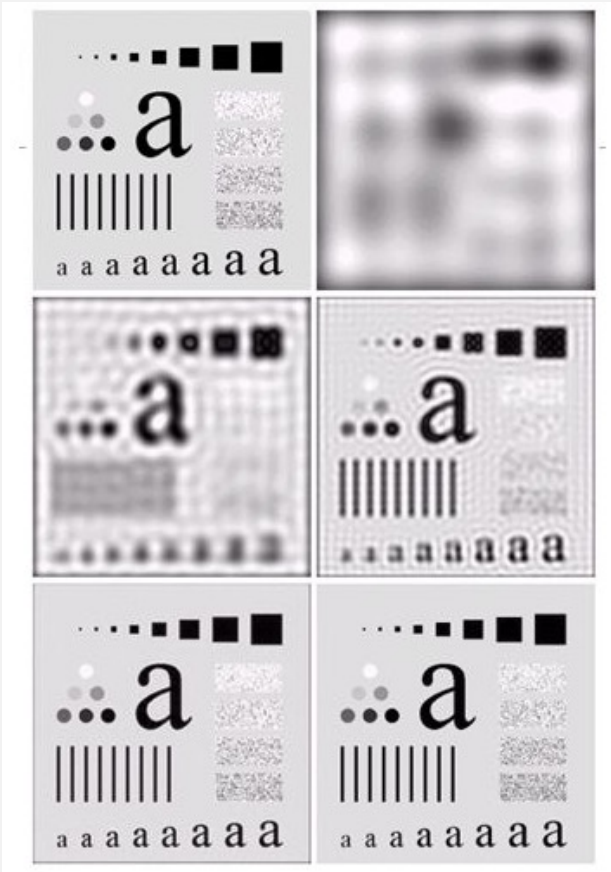


FIGURE 4.18 (a) Original image. (b)–(f) Results of filtering with Gaussian lowpass filters with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Figs. 4.12 and 4.15.

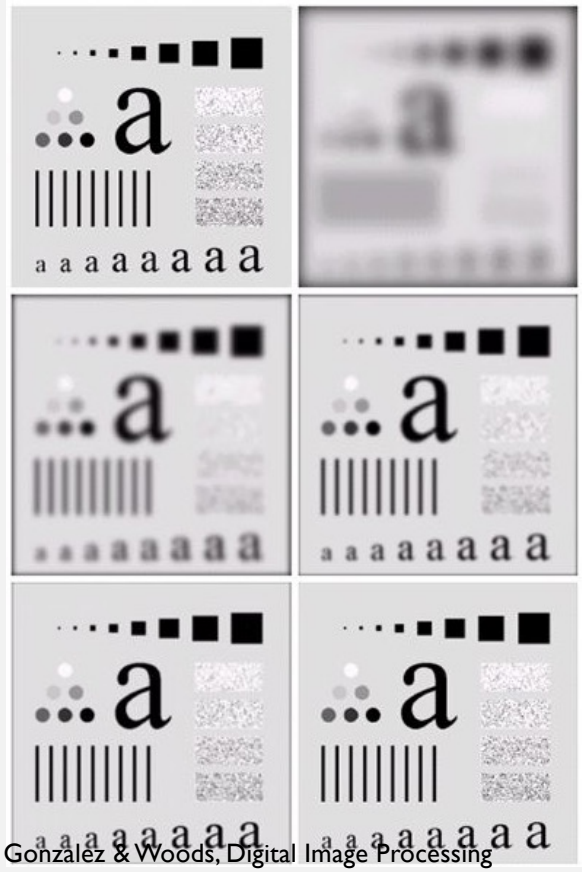
a b c d e f

GAUSSIAN LOW-PASS FILTERS

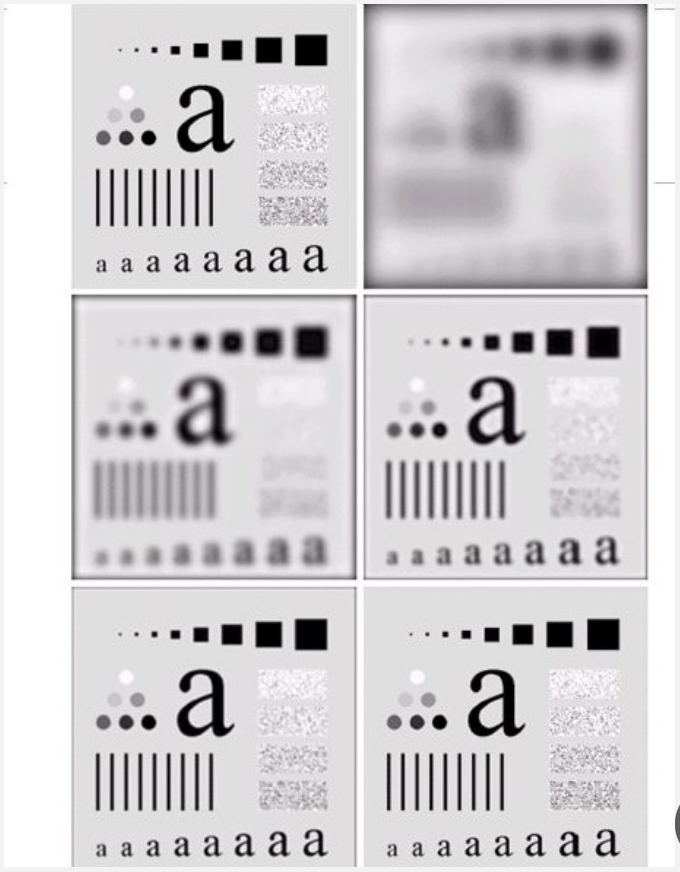
IDEAL LP FILTER



Gaussian LPF



Butterworth LPF



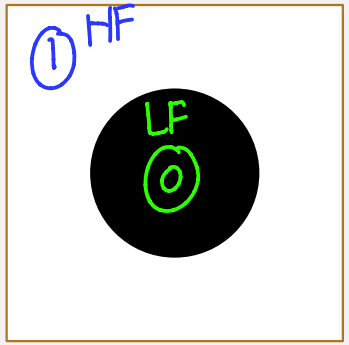
HIGH-PASS FILTERS

- High-pass filter - Sharpening Frequency-Domain Filters
- Transfer function of the high-pass filters:

$$H_{hp}(u,v) = 1 - H_{lp}(u,v)$$

- $H_{lp}(u,v)$: transfer function of the low-pass filters.

low pass filter



HIGH-PASS FILTERS

LOW-PASS

- Ideal high-pass filters, transfer function:

$$H(u,v) = \begin{cases} 0 & if\ D(u,v) \le D_0 \\ 1 & if\ D(u,v) > D_0 \end{cases}$$

$$H(u,v) = \begin{cases} 1 & if\ D(u,v) \le D_0 \\ 0 & if\ D(u,v) > D_0 \end{cases}$$

- Butterworth high-pass filters, transfer function:

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}}$$

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}$$

- Gaussian high-pass filters, transfer function:

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

LPF

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

HIGH-PASS FILTERS (HP)

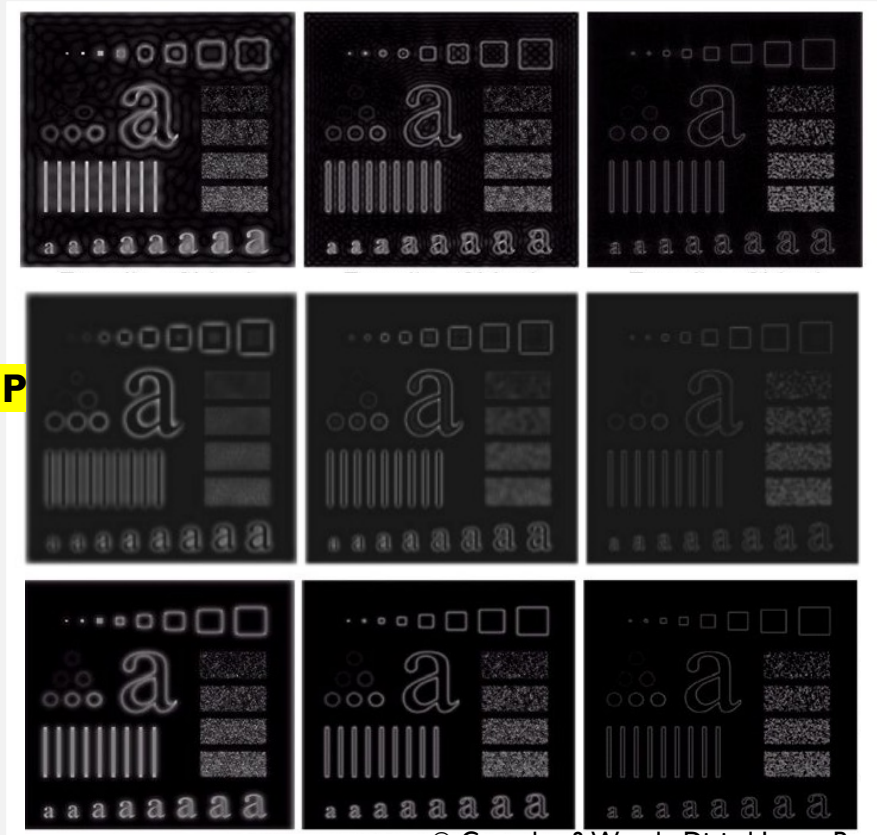
Edge

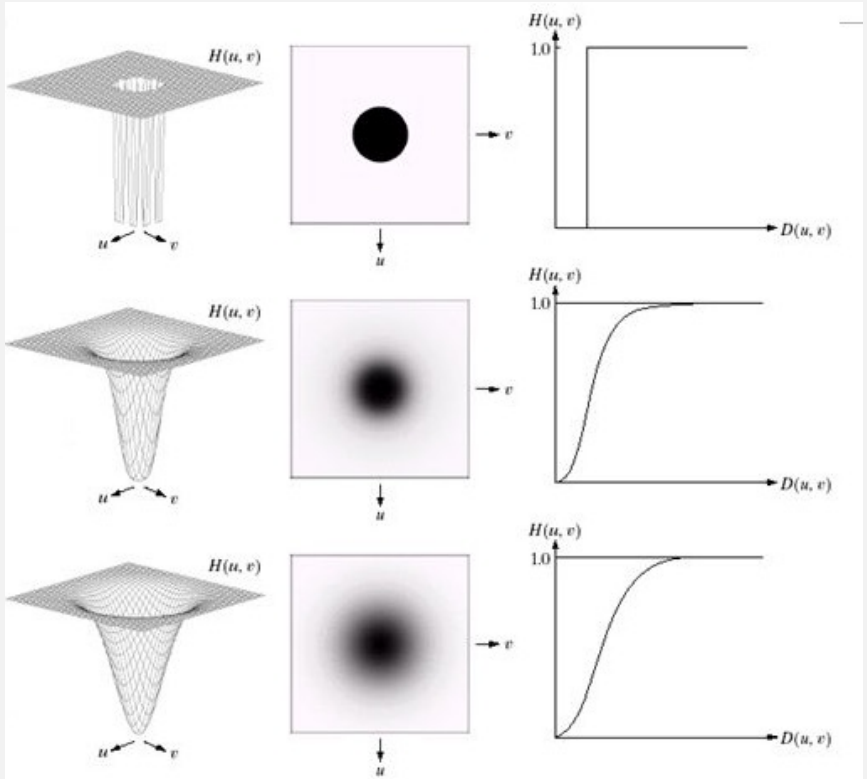
D_0=15 D_0=30 D_0=80

Ideal HP

Butterworth HP
Order = 2

Gaussian HP





a b c
d e f
g h i

FIGURE 4.22 Top row: Perspective plot, image representation, and cross section of a typical ideal highpass filter. Middle and bottom rows: The same sequence for typical Butterworth and Gaussian highpass filters.

Filter

CONVOLUTION THEOREM

- $F(u,v)$ and $H(u,v)$ are the Fourier transforms of $f(x,y)$ and $h(x,y)$
- $f(x,y) * h(x,y)$ and $F(u,v)H(u,v)$: Fourier transform pair:

Convo multiply

$$f(x,y) * h(x,y) \leftrightarrow F(u,v)H(u,v)$$

- $\leftrightarrow$: $f(x,y) * h(x,y)$ can be obtained from inverse Fourier transform of $F(u,v)H(u,v)$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- Convolution (spatial)

$$f(x,y) * h(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) h(x-m, y-n)$$

where $f(x,y)$ is an M x N image, $h(x,y)$ is a filter or kernel (impulse response) of an operation.

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Filtering**

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

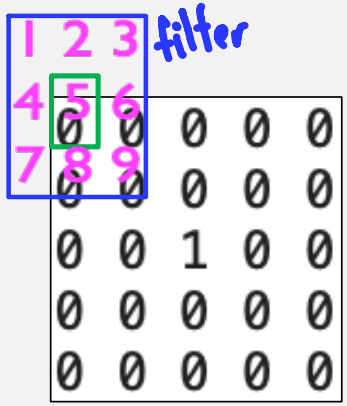
kernel h(x,y)

1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	9	8	7	0
0	6	5	4	0
0	3	2	1	0
0	0	0	0	0

Sliding the kernel over the image and find sum of products between image and kernel.



$$g(0,0) = h(-1,-1)f(-1,-1) + h(-1,0)f(-1,0) + h(-1,1)f(-1,1)$$
$$+h(0,-1)f(0,-1) + \boldsymbol{h(0,0)f(0,0)} + h(0,1)f(0,1)$$
$$+h(1,-1)f(1,-1) + h(1,0)f(1,0) + h(1,1)f(1,1)$$
$$= 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 0 + \mathbf{5 \cdot 0} + 6 \cdot 0 + 7 \cdot 0 + 8 \cdot 0 + 9 \cdot 0 = 0$$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Filtering**

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

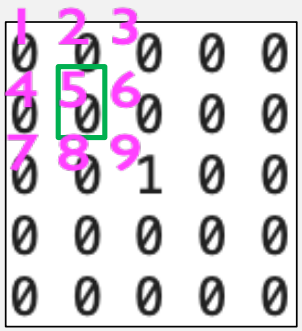
kernel h(x,y)

1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	9	8	7	0
0	6	5	4	0
0	3	2	1	0
0	0	0	0	0

Sliding the kernel over the image and find sum of products between image and kernel.



$$g(0,1) = g(0,2) = g(0,3) = g(0,4) = g(1,0) = g(2,0) = g(3,0) = g(4,0) = g(4,1) = g(4,2) = g(4,3) = g(4,4) = g(1,4) = g(2,4) = g(3,4) = 0$$

$$g(1,1) = 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 + 7 \cdot 0 + 8 \cdot 0 + 9 \cdot 1 = 9$$

$$g(1,2) = 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 + 7 \cdot 0 + 8 \cdot 1 + 9 \cdot 0 = 8$$

$$g(1,3) = 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 + 7 \cdot 1 + 8 \cdot 0 + 9 \cdot 0 = 7$$

$$g(2,1) = 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 1 + 7 \cdot 0 + 8 \cdot 0 + 9 \cdot 0 = 6$$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Convolution** – very similar to spatial filtering. Only the filter (transfer function) is flipped (mirrored) as derived from the linear system.

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

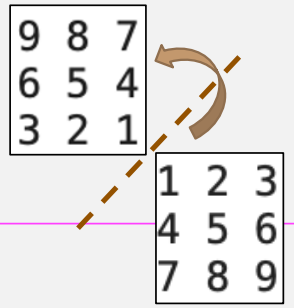
kernel h(x,y)

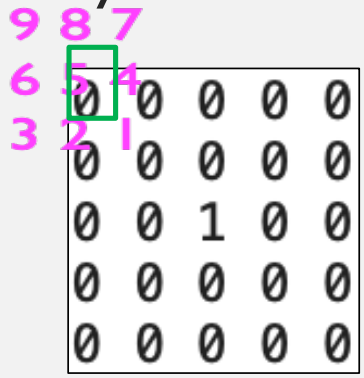
1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

1) Flipping the kernel
2) Sliding the kernel over the image and find sum of products between image and kernel.





$g(0,0) = g(0,0) = h(1,1)f(-1,-1) + h(1,0)f(-1,0) + h(1,-1)f(-1,1) + h(0,1)f(0,-1) + \boldsymbol{h(0,0)f(0,0)} + h(0,-1)f(0,1) + h(-1,1)f(1,-1) + h(-1,0)f(1,0) + h(-1,-1)f(1,1)$

$= 9 \cdot 0 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 0 + \mathbf{5 \cdot 0} + 4 \cdot 0 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 = 0$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Convolution** – very similar to spatial filtering. Only the filter (transfer function) is flipped (mirrored) as derived from the linear system.

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

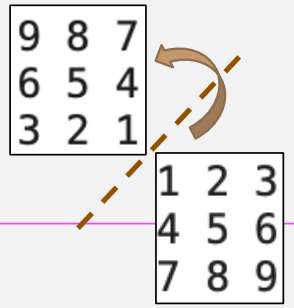
kernel h(x,y)

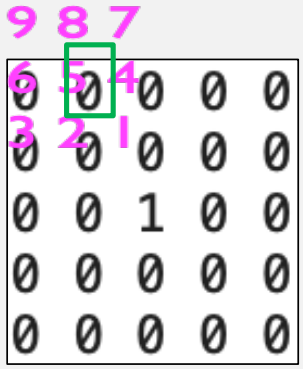
1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

1) Flipping the kernel
2) Sliding the kernel over the image and find sum of products between image and kernel.





$$g(0,1) = g(0,2) = g(0,3) = g(0,4)$$
$$= g(1,0) = g(2,0) = g(3,0) = g(4,0)$$
$$= g(4,1) = g(4,2) = g(4,3) = g(4,4)$$
$$= g(1,4) = g(2,4) = g(3,4) = 0$$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Convolution** – very similar to spatial filtering. Only the filter (transfer function) is flipped (mirrored) as derived from the linear system.

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

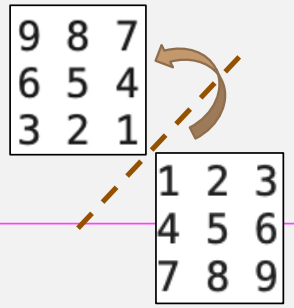
kernel h(x,y)

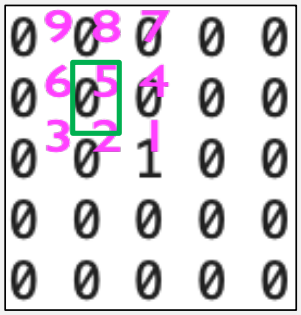
1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

1) Flipping the kernel
2) Sliding the kernel over the image and find sum of products between image and kernel.





$$g(1,1) = 9 \cdot 0 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 0 + \mathbf{5 \cdot 0} + 4 \cdot 0 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 1 = 1$$

$$g(1,2) = 9 \cdot 0 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 0 + \mathbf{5 \cdot 0} + 4 \cdot 0 + 3 \cdot 0 + 2 \cdot 1 + 1 \cdot 0 = 2$$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Convolution** – very similar to spatial filtering. Only the filter (transfer function) is flipped (mirrored) as derived from the linear system.

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

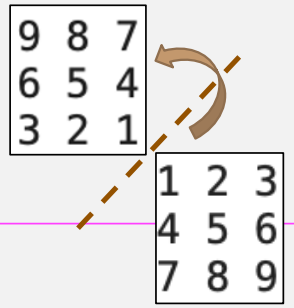
kernel h(x,y)

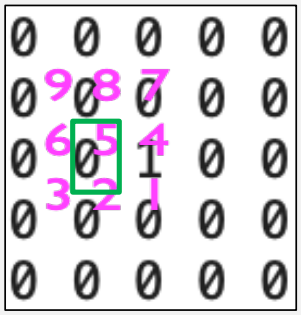
1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

1) Flipping the kernel
2) Sliding the kernel over the image and find sum of products between image and kernel.





$$g(2,1) = 9 \cdot 0 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 0 + \mathbf{5 \cdot 0} + 4 \cdot 1 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 = 4$$

$$g(2,2) = 9 \cdot 0 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 0 + \mathbf{5 \cdot 1} + 4 \cdot 0 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 = 5$$

SPATIAL FILTERING VS. FREQUENCY FILTERING

- **Convolution** – very similar to spatial filtering. Only the filter (transfer function) is flipped (mirrored) as derived from the linear system.

Image f(x,y)

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

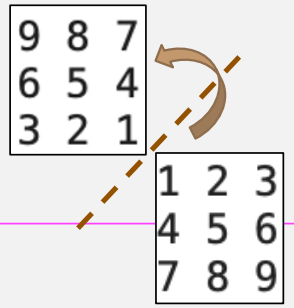
kernel h(x,y)

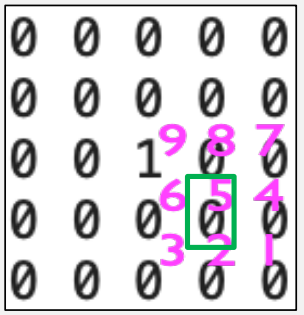
1	2	3
4	5	6
7	8	9

output $g(x, y)$

0	0	0	0	0
0	1	2	3	0
0	4	5	6	0
0	7	8	9	0
0	0	0	0	0

1) Flipping the kernel
2) Sliding the kernel over the image and find sum of products between image and kernel.

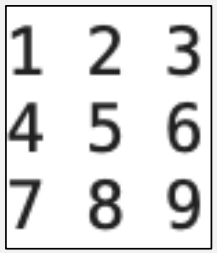




$$g(3,3) = 9 \cdot 1 + 8 \cdot 0 + 7 \cdot 0 + 6 \cdot 0 + 5 \cdot 0 + 4 \cdot 0 + 3 \cdot 0 + 2 \cdot 0 + 1 \cdot 0 = 9$$

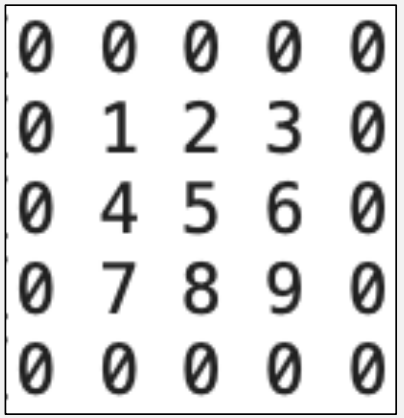
SPATIAL FILTERING VS. FREQUENCY FILTERING

image



kernel

after filtering



after convolution

Convolution – output is similar to the kernel.
Filtering – output is the correlation of the input and the corresponding kernel.

CONVOLUTION THEOREM

- $F(u,v)$ and $H(u,v)$ are the Fourier transforms of $f(x,y)$ and $h(x,y)$
- $f(x,y) * h(x,y)$ and $F(u,v)H(u,v)$: Fourier transform pair:

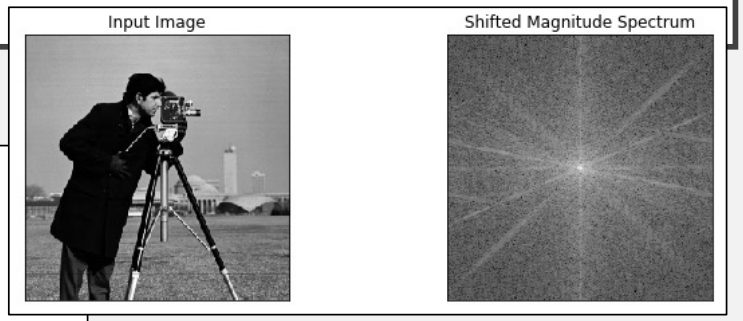
$$f(x,y) * h(x,y) \leftrightarrow F(u,v)H(u,v)$$

- $\leftrightarrow$: $f(x,y) * h(x,y)$ can be obtained from inverse Fourier transform of $F(u,v)H(u,v)$

LPF = Blur/smoothen → average filter

HPF = Edge → sharpening

PYTHON: IDEAL FILTERING



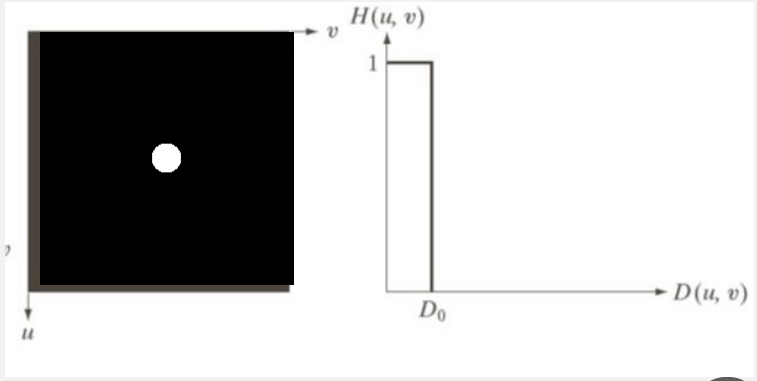
- Show Fourier transform:

```
img = cv2.imread('cameraman.tif',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = np.log(1+np.abs(fshift))
plt.imshow(img, cmap = 'gray')plt.title('Input Image')
plt.imshow(magnitude_spectrum.astype(np.uint8), cmap = 'gray')
plt.title('Shifted Magnitude Spectrum‘)
```

$$H(u,v) = \begin{cases} 1 & if\ D(u,v) \le D_0 \\ 0 & if\ D(u,v) > D_0 \end{cases}$$

- Create circle mask (low-pass filters):

```
nx, ny = (256, 256)
x = np.linspace(-nx/2, nx/2-1, nx)
y = np.linspace(-ny/2, ny/2-1, ny)
ideal_filter = np.ones((nx,ny),dtype=np.bool)
i, j = np.meshgrid(x, y, sparse=True)
ideal_filter = np.ones((nx,ny),dtype=np.bool)
ideal_filter = np.sqrt(i*i+j*j)<15
```

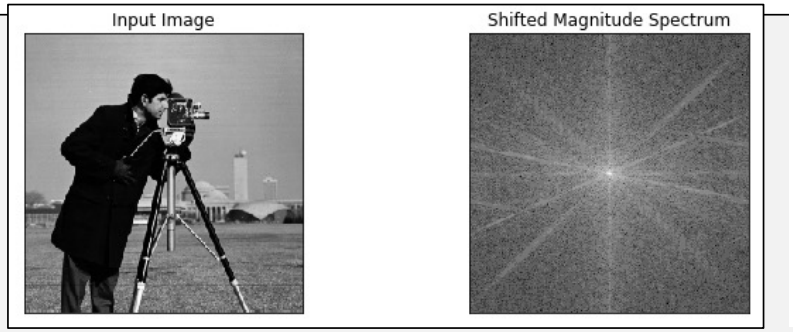


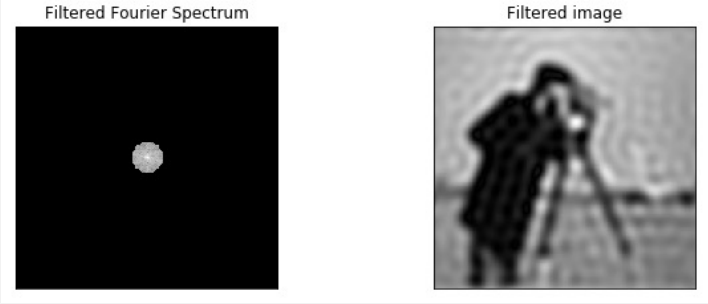
- **High-pass filters**: change c<15 to c>=15

PYTHON: IDEAL FILTERING

- Filtered by mask size **c = 15** and then compute **inverse** Fourier transform to view the image in spatial domain:

```
img_ft_filter = fshift * ideal_filter
img_ft_filter_spectrum = np.log(1+np.abs(img_ft_filter))
plt.imshow(img_ft_filter_spectrum.astype(np.uint8), cmap = 'gray')
plt.title('Filtered Fourier Spectrum')

f_ishift = np.fft.ifftshift(img_ft_filter)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)
plt.imshow(img_back.astype(np.uint8), cmap = 'gray')
plt.title('Filtered image');plt.show()
```

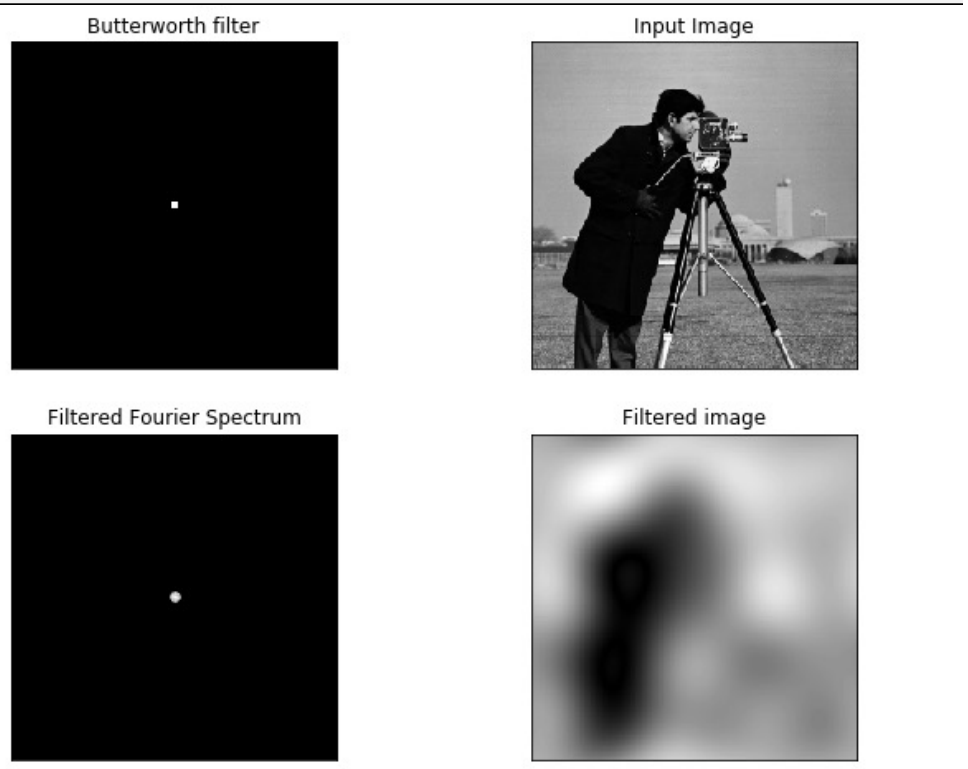




EXERCISE #3.1 : BUTTERWORTH FILTERING

- Low-pass filtering (Butterworth)

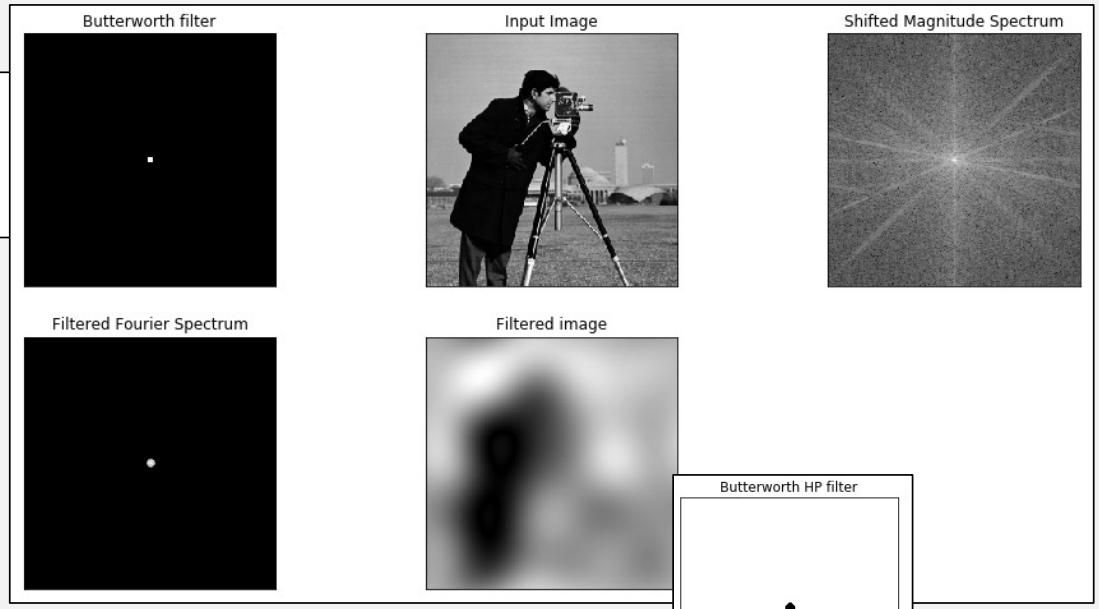
```
c = 15 # cutoff freq.
n = 15 # order

nx, ny = (256, 256)
x = np.linspace(-nx/2, nx/2-1, nx)
y = np.linspace(-ny/2, ny/2-1, ny)
btw = np.zeros((nx,ny),dtype=np.float32)
i, j = np.meshgrid(x, y, sparse=True)
btw = 1/(1+((i**2+j**2)/c)**(2*n))
```



EXERCISE #3.1 : BUTTERWORTH FILTERING

- Low-pass filtering (Butterworth) and answer what happen when increase c

```
img_ft_filter = fshift * btw
img_ft_filter_spectrum =
np.log(1+np.abs(img_ft_filter))
```



HIGH-PASS : 1- btw

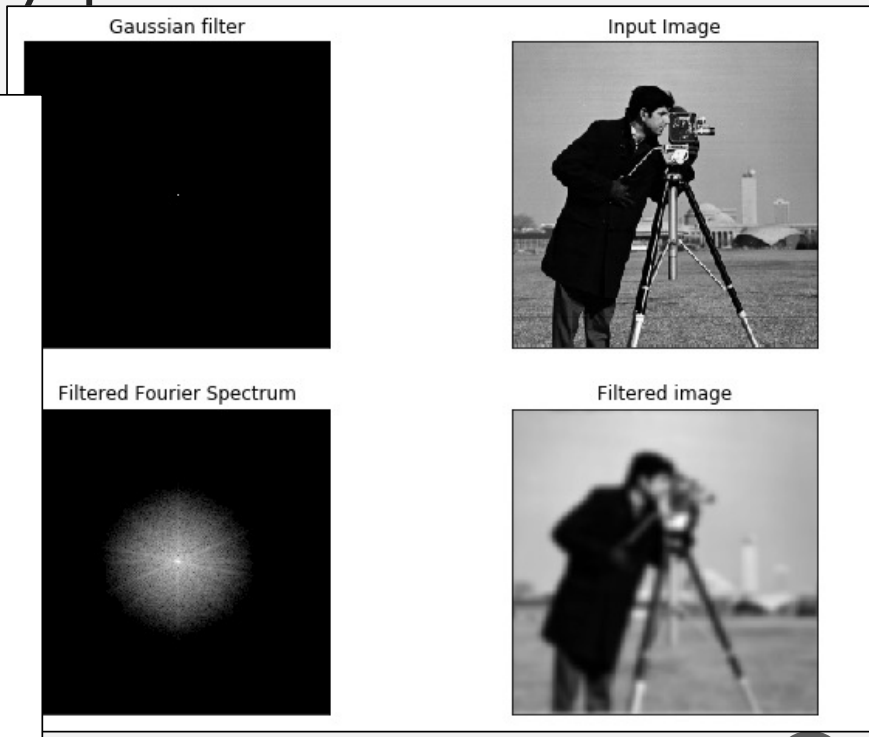
EXERCISE # 3.2 LOW-PASS GAUSSIAN FILTERING

- Complete the code below for program to apply lp Gaussian filtering on a 'cameraman.tif' image

```
c = 15 # cutoff freq.


nx, ny = (256, 256)
x = np.linspace(-nx/2, nx/2-1, nx)
y = np.linspace(-ny/2, ny/2-1, ny)
```

```

```

```
img_ft_filter = fshift * gaussian_filter
img_ft_filter_spectrum =
np.log(1+np.abs(img_ft_filter))
```



EXERCISE #3.3 APPLY HPF AND COMPARE WITH LPF

- Compare the results from **high-pass** filtering using either Ideal, Butterworth or Gaussian Filter using
  c = 15 (order n =15) and answer changes when modify c to be higher?

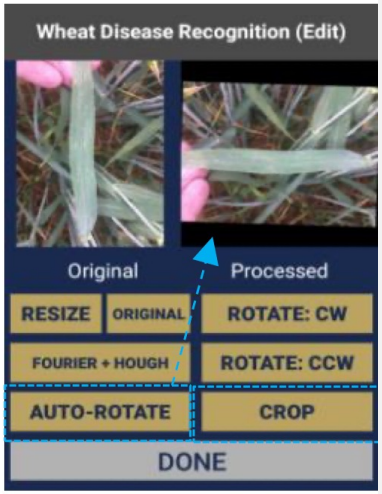
EXERCISE #4 LPF IN SPATIAL DOMAIN

- 4.1 Select one of the low pass filter and convert it back to time domain and see what it looks like?
- 4.2 Use Laplacian kernel to enhance the moon in frequency domain

FURTHER FOURIER TRANSFORM APPLICATIONS

- Applications of FT
  - Image enhancement
  - Image compression
  - Machine learning – cancer diagnosis
- Fast Fourier transform for fingerprint enhancement
  - Fast Fourier Transform of each homogeneous block is computed; the magnitude of the FFT is taken and raised to some real value *k*.
- FFT for pre-processing an image
  - Rotating an image for extract rotation-variant features

A.A. Amina, M. Dominique, Z. Youcef, Fast Fourier transform for fingerprint enhancement and features extraction with adaptive block size using orientation fields, *ITM Web of Conferences 43, 01014 (2022)*



<< Autorotated wheat leaf Using Fourier transform

P. Siricharoen, et al., A Lightweight Mobile System for Crop Disease Diagnosis, International Conference on Image Analysis and Recognition, 2016

FURTHER FOURIER TRANSFORM APPLICATIONS

- Encryption – watermasking
  - After the calculation of a DFT, the substitution of the bits to be integrated has been performed by modifying the LSBs of the quantized coefficients. This is performed by comparing the parity of the successive coefficients.
  - Watermark was integrated using the mid-band coefficients (mixed time and frequency components of the signal) which allowed us to guarantee imperceptibility while being robust to compression attacks and filtering.
- Compression
  - JPEG image compression algorithms

Kahlessenane Fares, Khaldi Amine, Euschi Salah. A robust blind color image watermarking based on Fourier T transform domain, *Optik*, 2020

SPATIAL FILTERING VS. FREQUENCY MULTIPLICATION

- Box filter size of 8000 x 8000
  - Spatial filtering uses average time : 13.78 s
  - Frequency multiplication uses average time : 9.03 s
    - Including fourier transform of image and filter, multiplication, inverse fourier transform of the resulted image

REFERENCES

- Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, Addison- Wesley
  - Chapter 4 – Image Enhancement in the Frequency Domain