Assignment -3

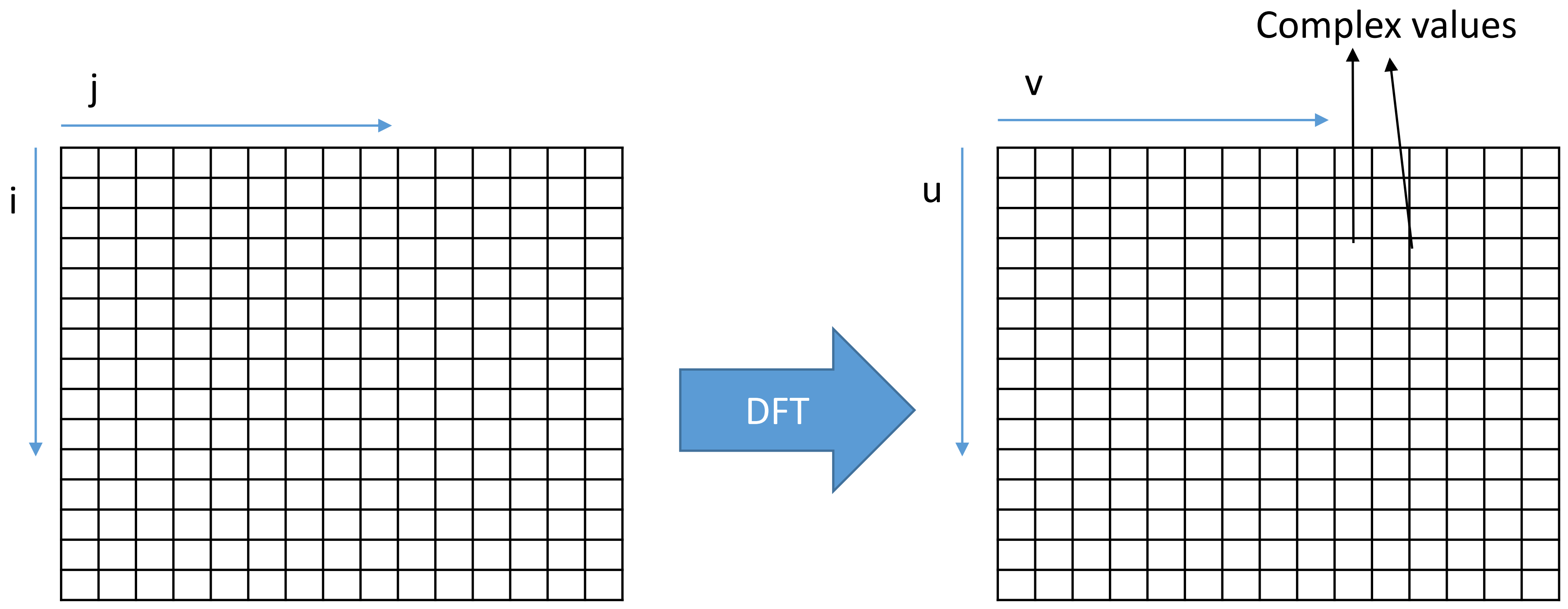
1. DFT Implementation
    a. Forward Fourier transform
    b. Inverse Fourier transform
    c. Magnitude of DFT
2. Spatial Filtering
    a. Using Gaussian 5X5 filter
    b. Using Laplacian 3X3 filter
3. Frequency filtering
    a. Reject noise frequencies by analyzing the DFT

Due Date: Nov. 20th , 11:59 PM

Part 1: DFT Implementation

1. Input a matrix of integers (15X15)
   1. Compute forward Fourier transform
   2. Compute inverse Fourier transform
   3. Compute magnitude of DFT

Forward Fourier transform



Input 15X15 matrix (I)

Output DFT 15X15 matrix (F)

Forward Fourier transform

$$F(u,v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j)\, e^{-\sqrt{-1}\frac{2\pi}{N}(ui+vj)}, N = 15$$

$$F(u,v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j)\, [\cos\left[\frac{2\pi}{N}(ui + vj)\right] - \underbrace{\sqrt{-1}}_{\text{In python, } \sqrt{-1} = 1j} \sin\left[\frac{2\pi}{N}(ui + vj)\right]]$$

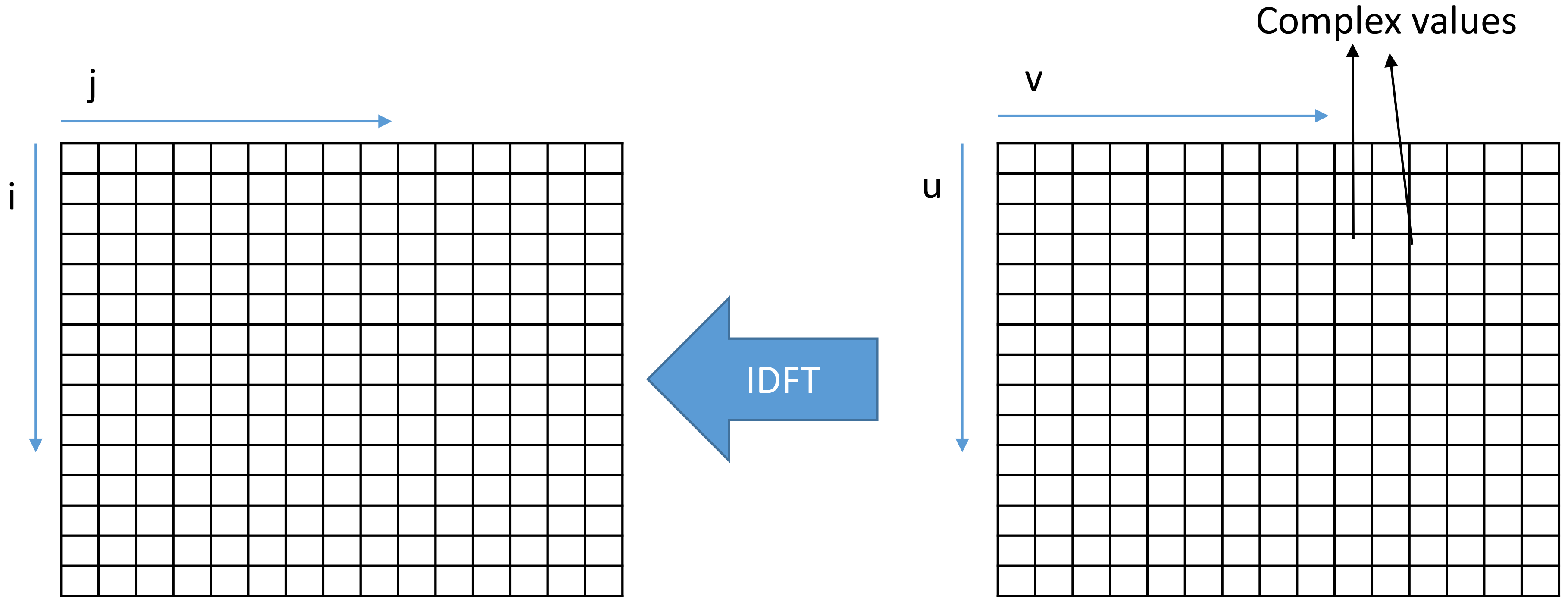
Forward Fourier transform

$$F(u,v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j)\, e^{-\sqrt{-1}\frac{2\pi}{N}(ui+vj)}, N = 15$$

$$F(u,v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j)\, [\cos\left[\frac{2\pi}{N}(ui+vj)\right] - \sqrt{-1}\sin\left[\frac{2\pi}{N}(ui+vj)\right]]$$

u = {0,..14}, v = {0,..14}

Inverse Fourier transform



Input 15X15 matrix (I)

Output DFT 15X15 matrix (F)

Inverse Fourier transform

$$I(i,j) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v)\, e^{\sqrt{-1}\frac{2\pi}{N}(ui+vj)}, N = 15$$

$$\underbrace{I(i,j)}_{\text{End up getting complex numbers}} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v)\, \left[\cos\left[\frac{2\pi}{N}(ui+vj)\right] + \sqrt{-1}\sin\left[\frac{2\pi}{N}(ui+vj)\right]\right]$$

i = {0,..14}, j = {0,..14}

Magnitude of DFT

$$F(u,v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i,j) \, [\cos \left[\frac{2\pi}{N} (ui + vj) \right] - \sqrt{-1} \sin \left[\frac{2\pi}{N} (ui + vj) \right]]$$

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

Please compute magnitude in the function. Do not use inbuilt function (such as abs).

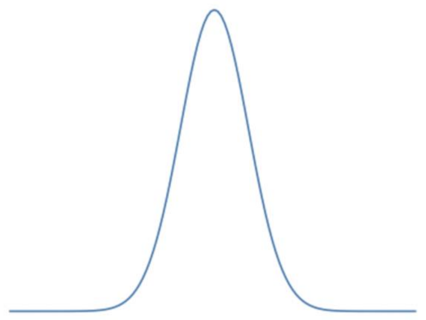
Part-2: Spatial Filtering

The input is an image, and string specifying the type of filter to use

1. Perform the required zero padding
2. Create the filter
    1. 5X5 Gaussian filter
    2. 3X3 Laplacian filter
3. Perform convolution

Gaussian Filter

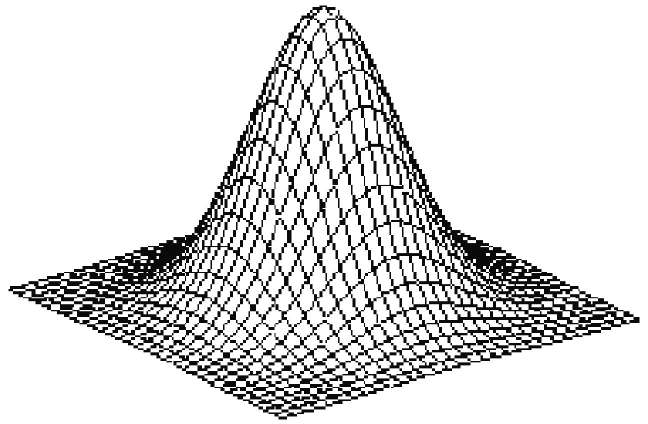
- Higher weightage for pixels in the middle.



¼ *

1	2	1

1D

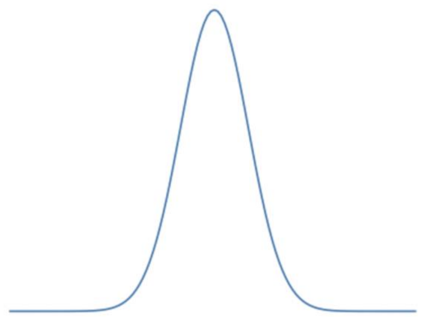


$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

You can design a 5X5 filter using the formula for a 2D Gaussian.
(x,y) -> co-ordinates where the center of the kernel is (0,0)
Do not forget the normalization factor.

Gaussian Filter

- Higher weightage for pixels in the middle.

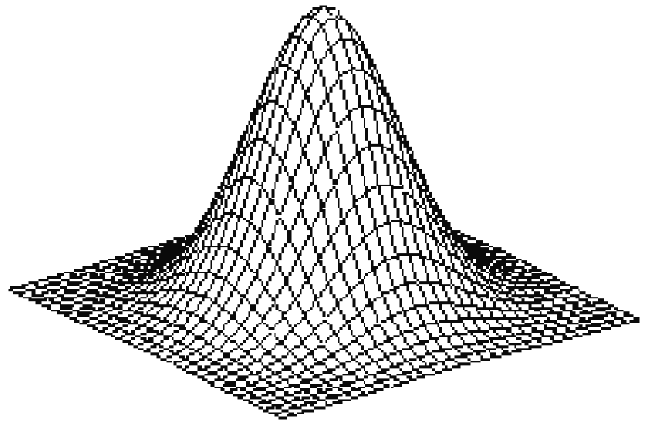


¼ *

1	2	1

1D

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



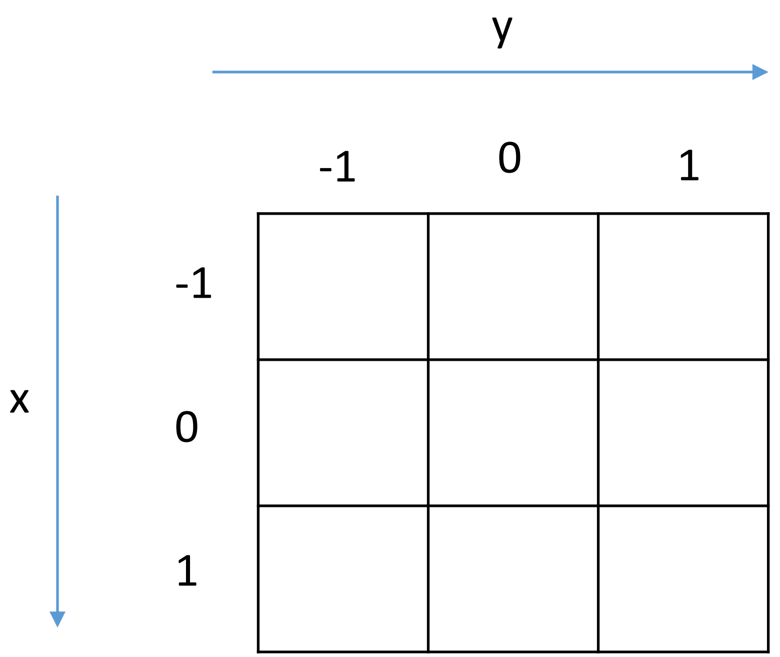
1/16 *

1	2	1
2	4	2
1	2	1

2D, 3X3

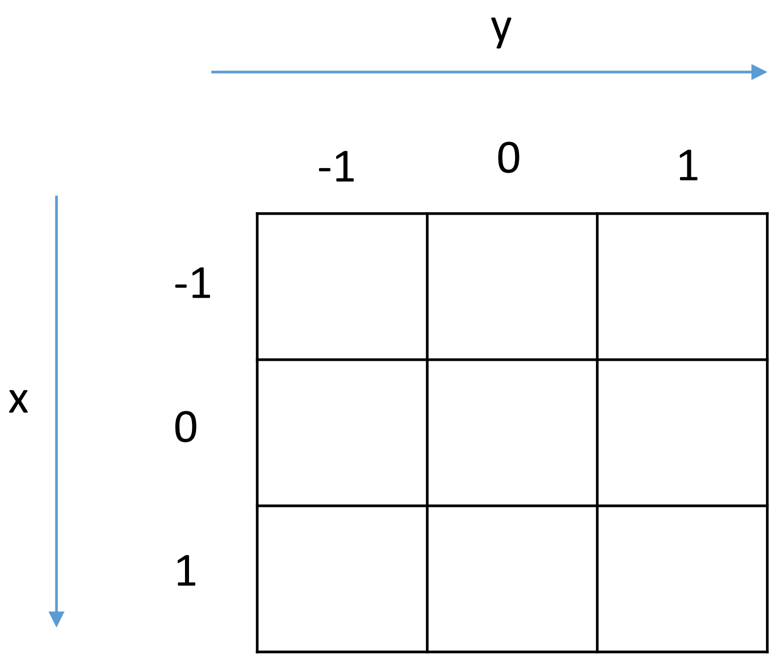
You can design a 5X5 filter using the formula for a 2D Gaussian.
Don't forget the normalization factor.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Don’t forget to normalize by factor f (so that the sum is 1)

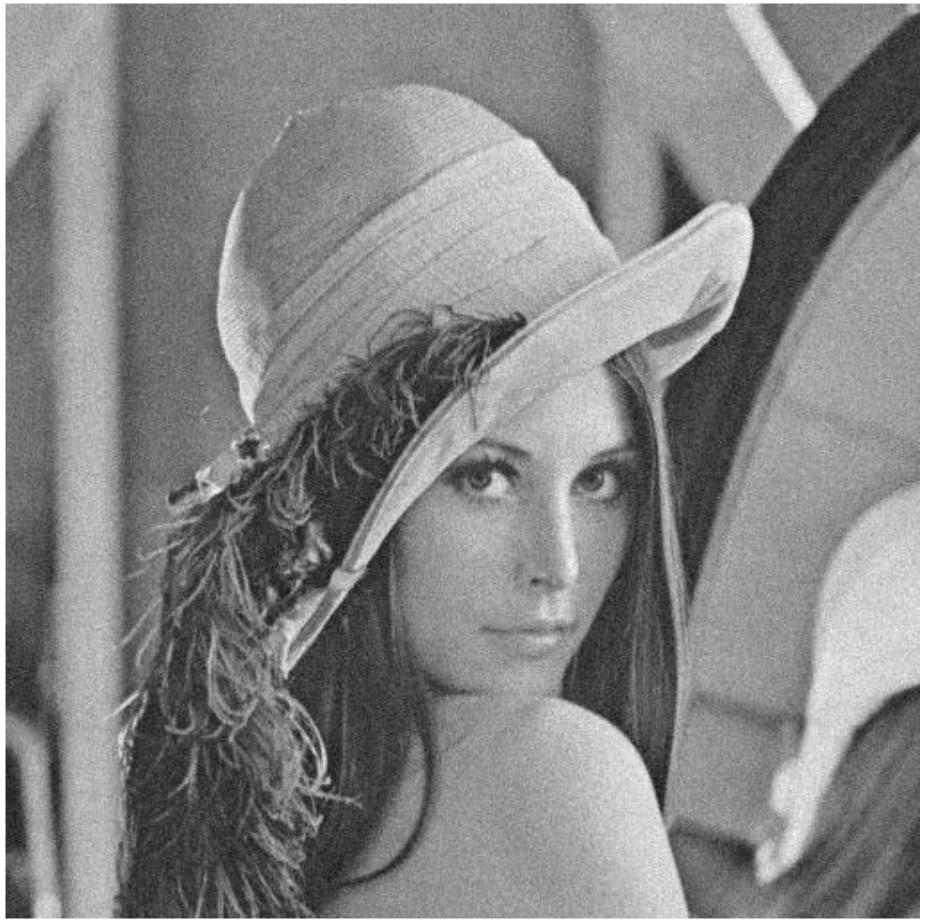
$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

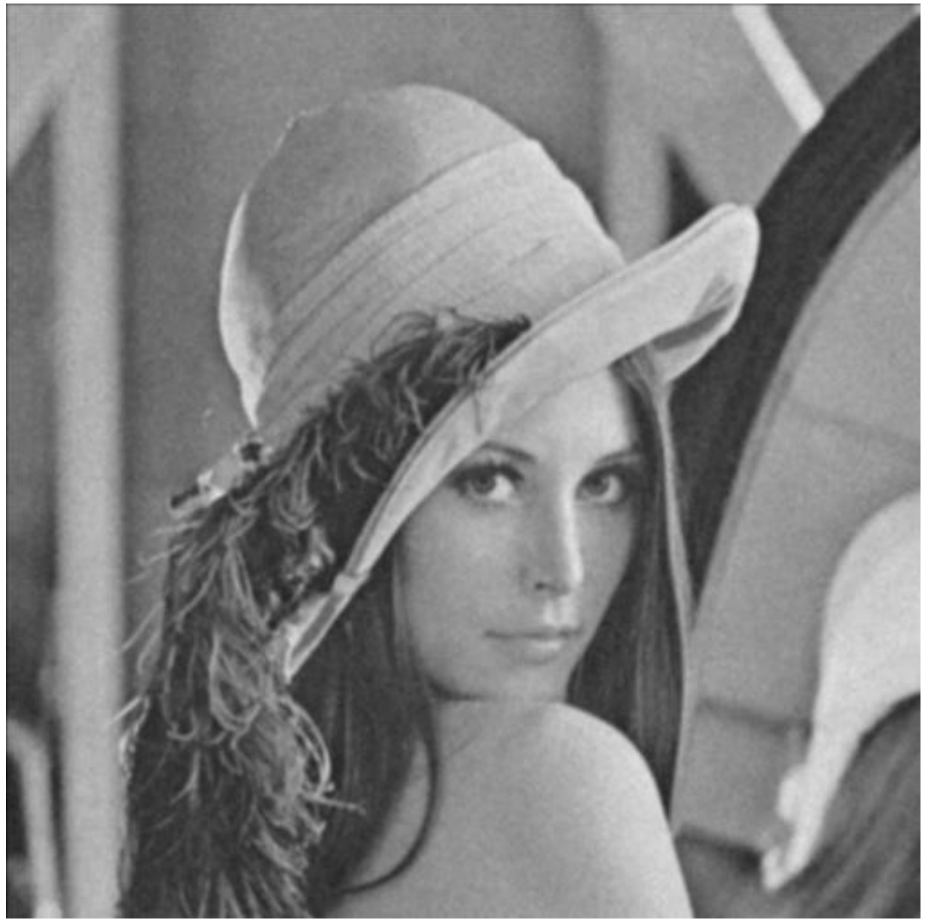


$$\frac{1}{16} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

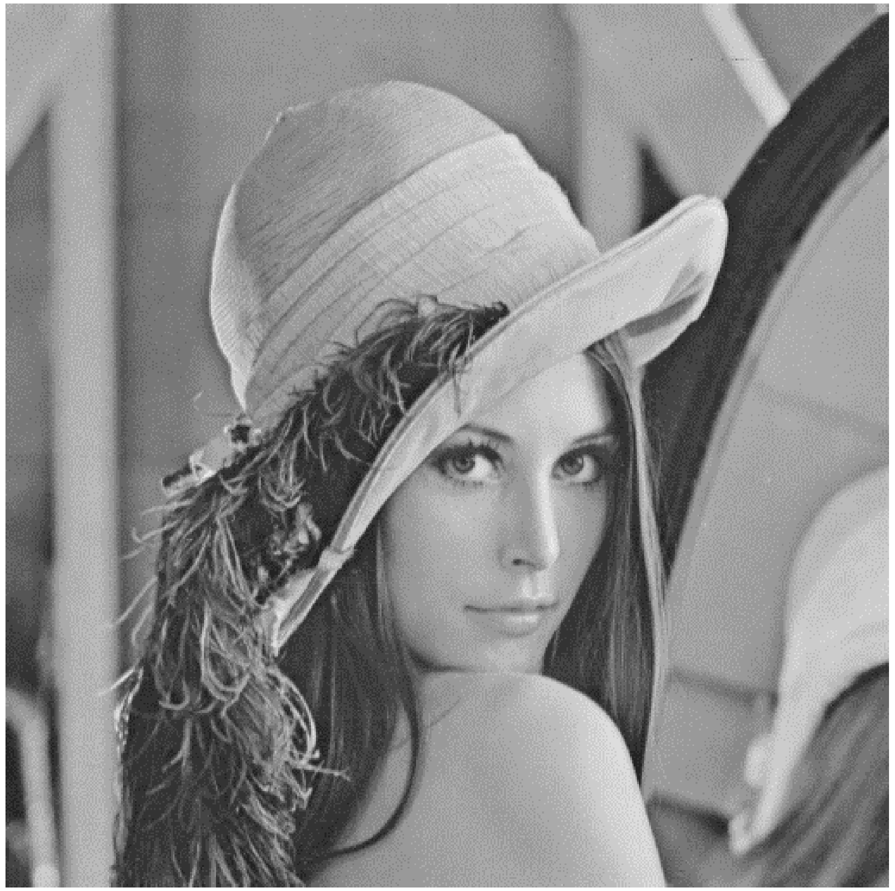
Don’t forget to normalize by factor f (so that the sum is 1)

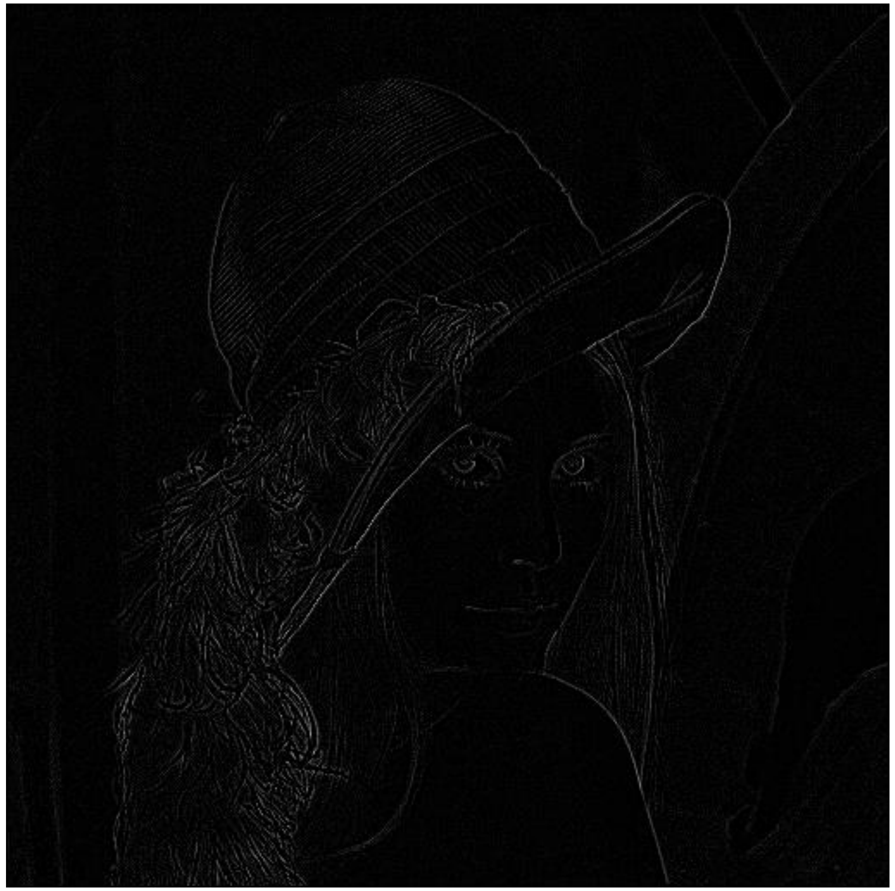
Gaussian Example


Input


Filtering result

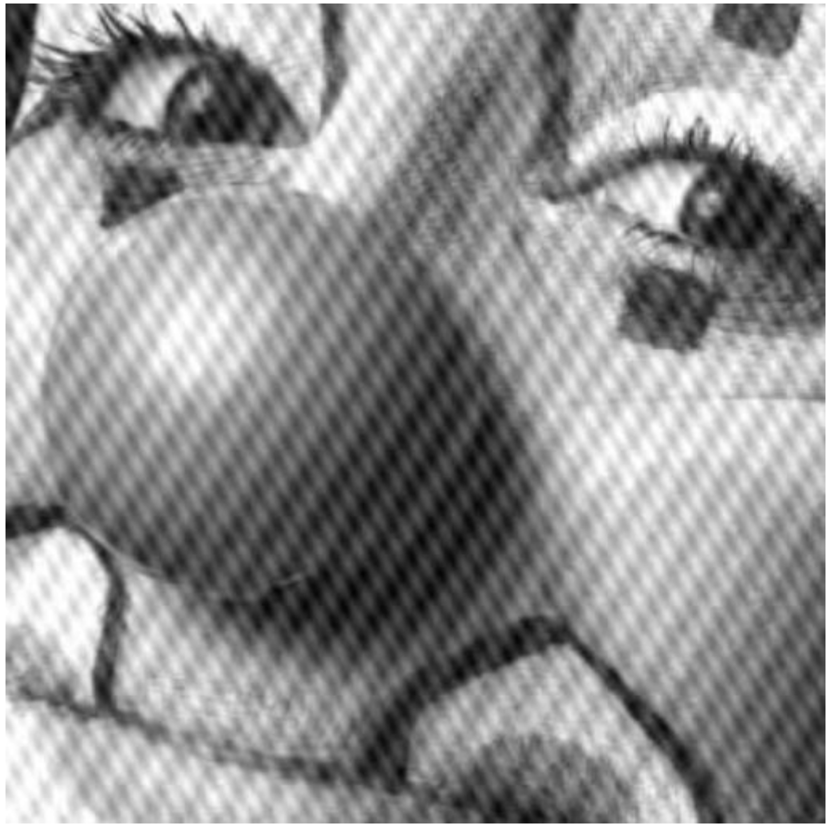
Laplacian Example


Input


Filtering result

Part 3: Frequency filtering

- Input Image



Steps

1. Compute the DFT

Computation of the DFT

- Fast algorithms for the DFT are collectively referred to as **Fast Fourier Transform** (FFT) algorithms.
- We will not delve into the design of these, as they are available in most math library programs.
  - Divide and conquer
  - Exploit Symmetry
- Reduces complexity from $O(n^2)$ to $O(n \log n)$

Forward Fourier transform

1. Compute the Fourier transform (numpy has *fft* and opencv both has *dft*)
2. Compute the shift (Ex. Numpy has *fftshift*)

Note: Numpy represents real part and imaginary part in a single matrix, however, opencv uses two matrices to represent the same.

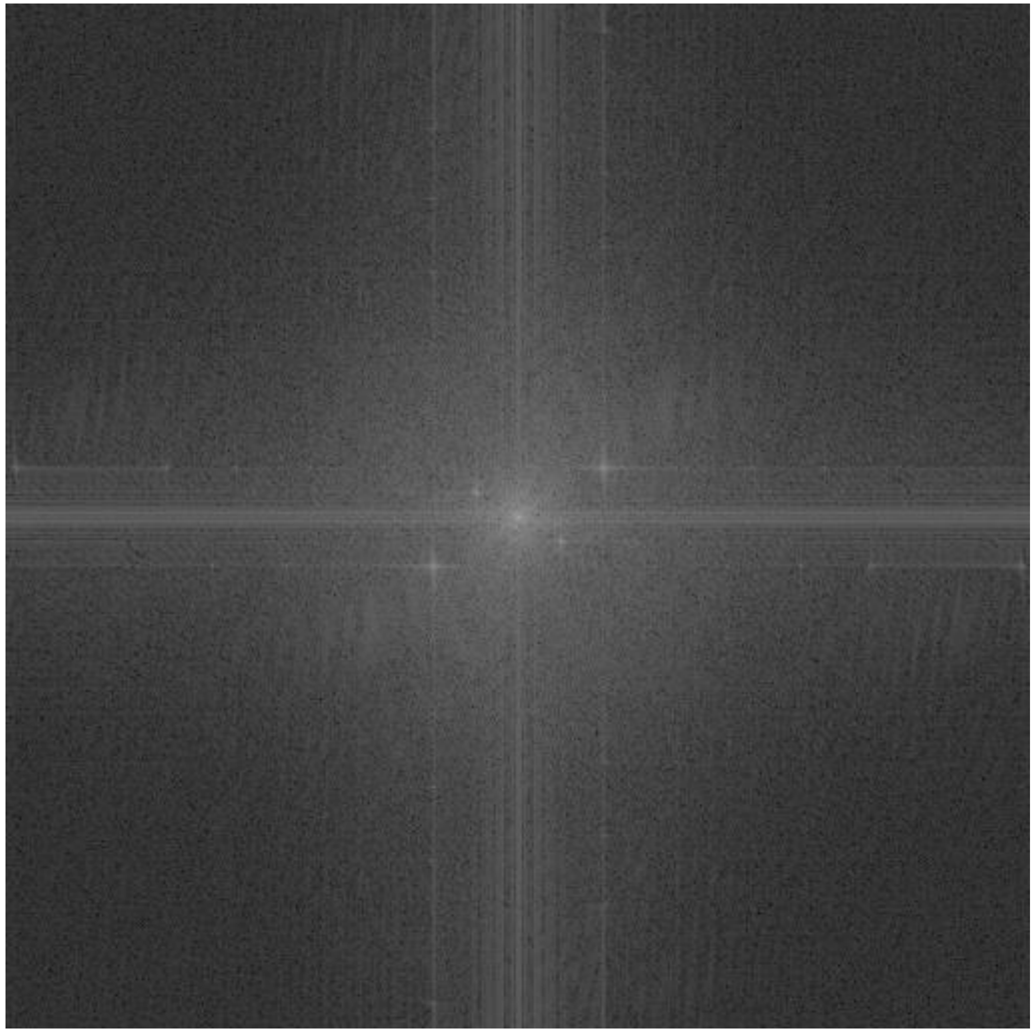
Displaying DFT

- DFT is one of the images that needs to be save.
- However the magnitude of the DFT could be large values
- In order to save as an image that is visible,

1. Do a logarithmic compression (Ex. *np.log(mag(DFT)*)
2. Convert to uint8 to save it as greyscale image

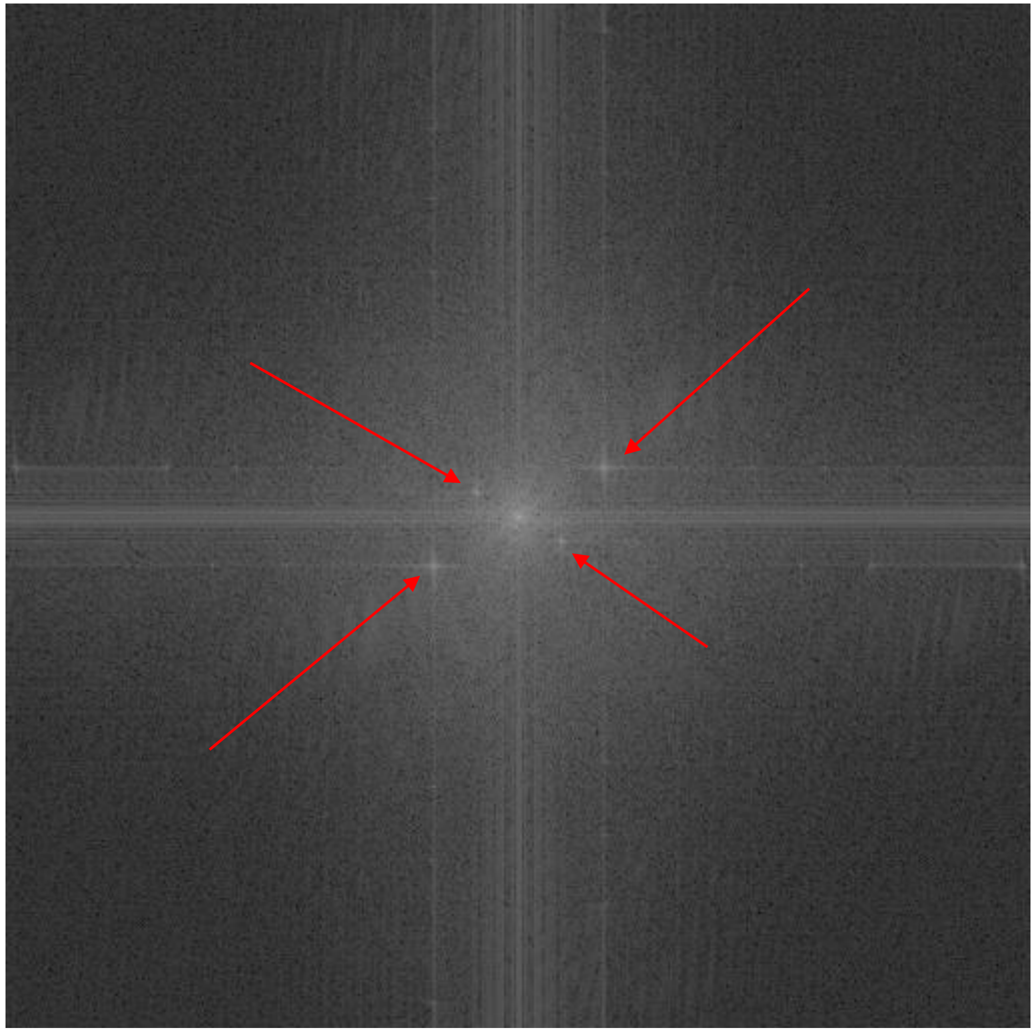
Steps

1. Compute the DFT



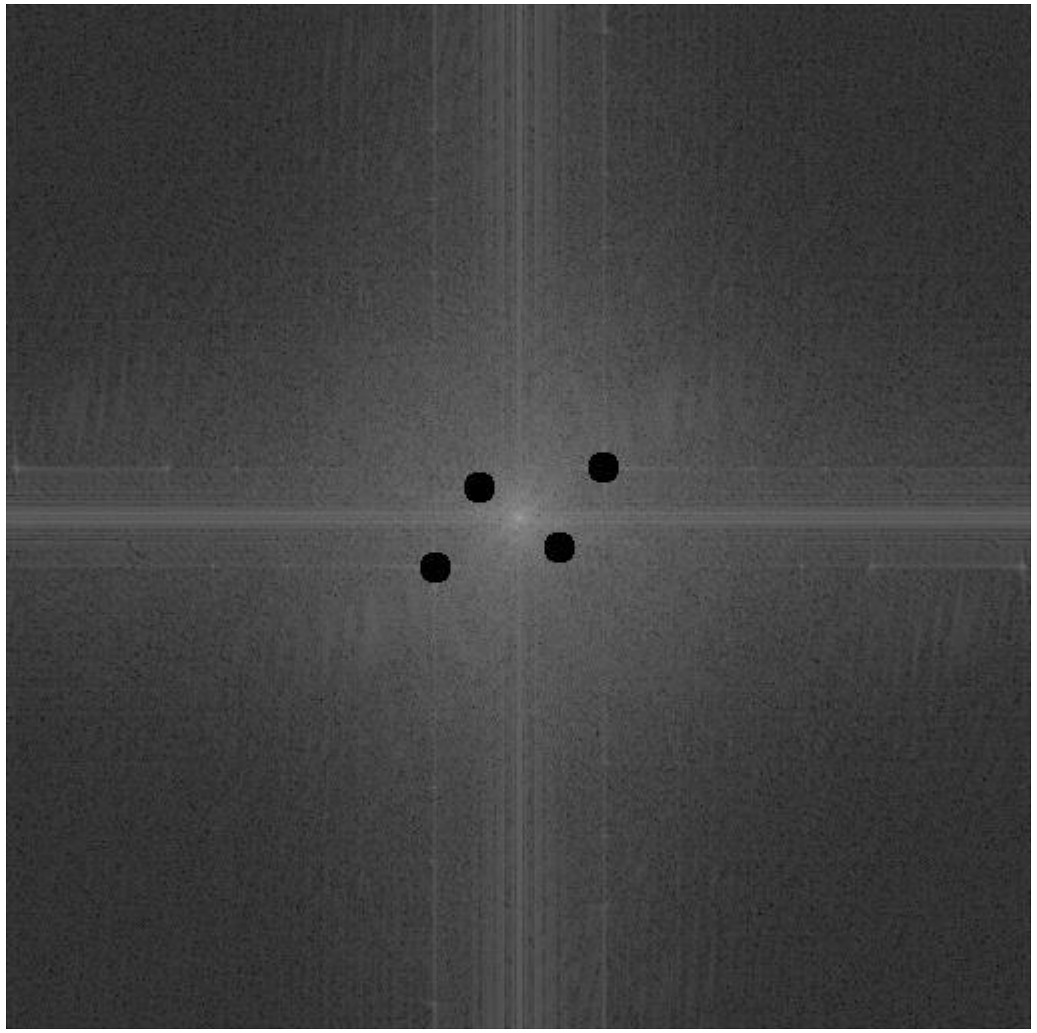
Steps

1. Compute the DFT
2. Inspect and Identify noise frequencies



Steps

1. Compute the DFT
2. Inspect and Identify noise frequencies
3. Create filter to reject noise frequencies



Filtering

- Filtered DFT is one of the images that needs to be saved.
- Follow same process as before (DFT image)

Steps

1. Compute the DFT
2. Inspect and Identify noise frequencies
3. Create filter to reject noise frequencies
4. Compute inverse Fourier Transform

Inverse Fourier transform

- Compute the inverse shift (Ex. Numpy fft library has *ifftshift*)
- Compute the inverse fourier transform (opencv has idft and numpy has ifft)

Filtered Image

- The inverse Fourier transform will give complex numbers.
- To transform into a meaningful image

1. Compute the magnitude
2. Do a full scale contrast stretch (to greyscale values)

Assignment -3

1. DFT Implementation(15 Pts)
2. Spatial filtering (30 Pts)
3. Frequency filtering (30 Pts)

Total: 75 Pts.

Submission Instructions

- Must use the **starter code** available in **Github**
- Submission allowed only through **Github**
- You will receive an email with invitation to join **Github** classroom
- Start by reading the **readme.md** file. Instructions are available here
- Github will **automatically** save the **last commit as a submission** before the deadline