Image Enhancement in Frequency Domain

Dr. Su Su Maung
CEIT Department
Yangon Technological University

Outline of Lecture5

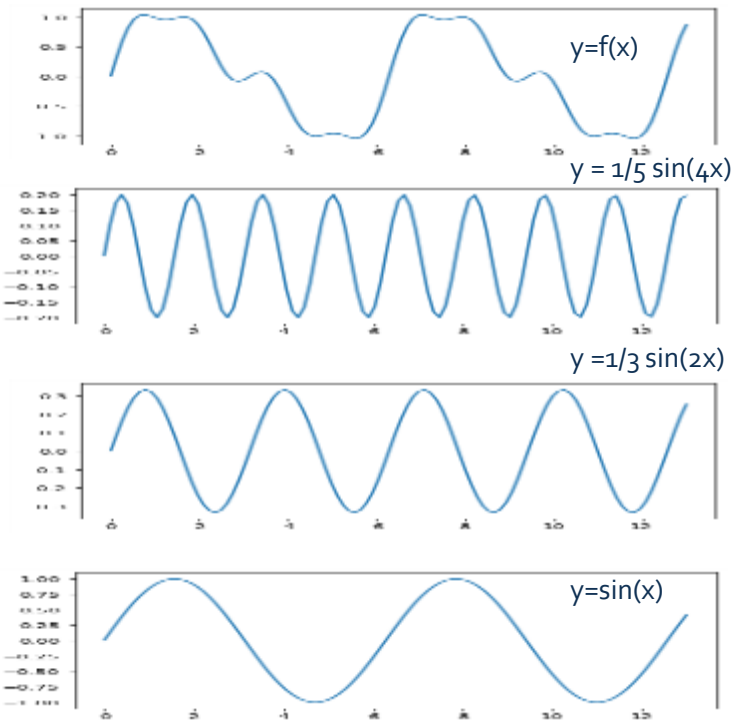
- Introduction
- The one-dimensional discrete Fourier transform(DFT)
- The two-dimensional DFT
- Some properties of the two dimensional Fourier transform
- Fourier transforms in Python
- Filtering in frequency domain

Introduction

- The Fourier Transform is of fundamental importance to image processing and allows to perform tasks more quickly which would be impossible to perform any other way.
- The Fourier Transform provides a powerful alternative to linear spatial filtering; it is more efficient to use the Fourier transform than a spatial filter for a large filter.
- The Fourier Transform also allows us to isolate and process particular image "frequencies", and so perform low-pass and high-pass filtering with a great degree of precision.

Background

- periodic function may be written as the sum of sines and cosines of varying amplitudes and frequencies.



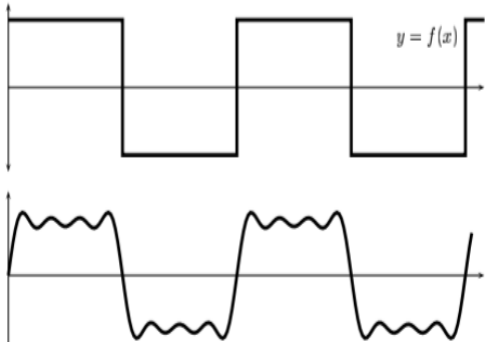
f(x) = sin(x)+1/3 sin(2x) + 1/5s in(4x)

Background

- For example, a "square wave", such as is shown in figure, has the decomposition

$$f(x) = sinx + \frac{1}{3}sin3x + \frac{1}{5}sin5x + \frac{1}{7}sin7x + \frac{1}{9}sin9x$$

- we take the first four terms only to provide the approximation. The more terms of the series we take, the closer the sum will approach the original function.



$f(x) = sinx + \frac{1}{3}sin3x + \frac{1}{5}sin5x + \frac{1}{7}sin7x + \frac{1}{9}sin9x$

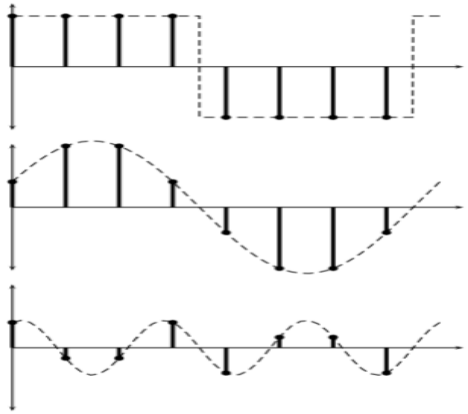
A square wave and its trigonometric approximation

The one-dimensional discrete Fourier transform

- When we deal with a discrete function, consider for example the discrete sequence

1, 1, 1, 1, -1, -1, -1, -1

- The Fourier transform allows us to obtain those individual sine waves which compose a given function or sequence.



Discrete function as the sum of sines

Definition of the one dimensional DFT

- For a sequence of length N

$$f = [f_0, f_1, f_2, \dots, f_{N-1,}]$$

- We define its discrete Fourier transform to be the sequence

$$F = [F_0, F_1, F_2, \dots, F_{N-1,}]$$

$$F_u = \sum_{x=0}^{N-1} \exp[-2\pi i \frac{xu}{N}] f_x$$

- Inverse DFT is very similar to the forward transform:

$$f_x = \frac{1}{N} \sum_{x=0}^{N-1} \exp[-2\pi i \frac{xu}{N}] F_u$$

The Fast Fourier Transform (FFT)

- Extremely fast and efficient algorithms for computing a DFT
- The use of an FFT vastly reduces the time needed to compute a DFT.

2^n	Direct arithmetic	FFT	Increase in speed
4	16	8	2.0
8	84	24	2. 67
16	256	64	4.0
32	1024	160	6.4
64	4096	384	10.67
128	16384	896	18.3
256	65536	2048	32.0
512	262144	4608	56.9
1024	1048576	10240	102.4

Comparison of FFT and direct arithmetic

The two-dimensional DFT

- In two dimensions, the DFT takes a matrix as input, and returns another matrix, of the same size, as output.
- If the original matrix values are *f*(x,y), where x and y are the indices, then the output matrix values are *F*(u,v).

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp[-2\pi i\left(\frac{xu}{M}+\frac{yu}{N}\right)]$$

- Then the original matrix *f* is the inverse Fourier transform of *F*

$$f(x,y) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} F(u,v)\exp[2\pi i\left(\frac{xu}{M}+\frac{yu}{N}\right)]$$

- Two-dimensional function f(x,y) can be expressed as sums of "corrugation" functions which have the general form

$$z = \mathrm{a}\sin(bx+cy)$$

Some properties of the two dimensional Fourier transform

- Similarity
- The DFT as a spatial filter
- Separability
- Linearity
- The convolution theorem
- The DC coefficient
- Shifting
- Conjugate symmetry

Some properties of the two dimensional Fourier transform

- **Similarity**
  - the forward and inverse transforms are very similar.

$$F_u = \sum_{x=0}^{N-1} \exp[-2\pi i \frac{xu}{N}] f_x$$

$$f_x = \frac{1}{N} \sum_{x=0}^{N-1} \exp[-2\pi i \frac{xu}{N}] F_u$$

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \exp[-2\pi i \left(\frac{xu}{M} + \frac{yu}{N}\right)]$$

$$f(x,y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F(u,v) \exp[2\pi i \left(\frac{xu}{M} + \frac{yu}{N}\right)]$$

Some properties of the two dimensional Fourier transform

- **The DFT as a spatial filter**
  - Note that the values $\exp[\pm 2\pi i(\frac{xu}{M}+\frac{yu}{N})]$ are independent of the values f or F .
  - This means that they can be calculated in advance, and only then put into the formulas above.
  - It also means that every value F(u,v) is obtained by multiplying every value of f(x,y) by a fixed value, and adding up all the results.
  - But this is precisely what a linear spatial filter does: it multiplies all elements under a mask with fixed values, and adds them all up.
  - Thus we can consider the DFT as a linear spatial filter which is as big as the image.

Some properties of the two dimensional Fourier transform

- An N-point DFT is expressed as the multiplication $F = Wf$ where f is the original input signal, W is the N-by-N square DFT matrix

$$W = \frac{1}{N}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

- where $\omega = \exp[\frac{-2\pi i}{N}]$

Some properties of the two dimensional Fourier transform

- f=[1,2,3,4] so that N=4. Then

$$\omega = \exp\left[\frac{-2\pi i}{4}\right] = -i$$

$$W = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & (-i)^2 & (-i)^3 \\ 1 & (-i)^2 & (-i)^4 & (-i)^6 \\ 1 & (-i)^3 & (-i)^6 & (-i)^9 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

$$F = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 10 \\ -2+2i \\ -2 \\ -2-2i \end{bmatrix}$$

Some properties of the two dimensional Fourier transform

- **Separability**
  - Notice that the Fourier transform "filter elements" can be expressed as products:

$$\exp\left[2\pi i\left(\frac{xu}{M}+\frac{yu}{N}\right)\right]=\exp\left[2\pi i\frac{xu}{M}\right]\exp\left[2\pi i\frac{yu}{N}\right]$$

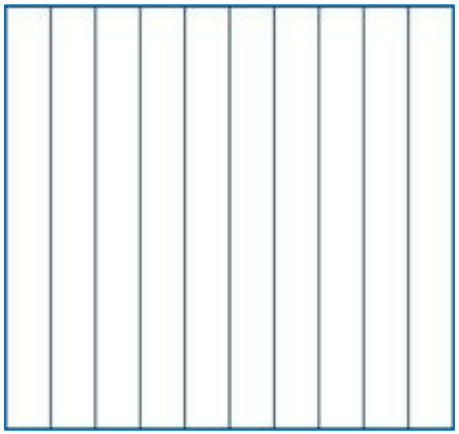
  - The first product value $\exp\left[2\pi i\frac{xu}{M}\right]$ depends only on x and u, and is independent of y and v.
  - Conversely, the second product value $\exp\left[2\pi i\frac{yu}{N}\right]$ depends only y on v and , and is independent of x and u. This means that we can break down our formulas above to simpler formulas that work on single rows or columns:
  - The 2-D DFT can be calculated by using this property of "separability"; to obtain the 2-D DFT of a matrix, we first calculate the DFT of all the rows, and then calculate the DFT of all the columns of the result,
  - Since a product is independent of the order, we can equally well calculate a 2-D DFT by calculating the DFT of all the columns first, then calculating the DFT of all the rows of the result.

Some properties of the two dimensional Fourier transform

- **Separability**

Original image

DFT of each row



DFT of each column

Calculating a 2D DFT

Some properties of the two dimensional Fourier transform

- Linearity
  - An important property of the DFT is its linearity; the DFT of a sum is equal to the sum of the individual DFT's, and the same goes for scalar multiplication:

$$F(f+g) = F(f) + F(g)$$

  - where k is a scalar, f and g and are matrices.

$$F(kf) = kF(f)$$

Some properties of the two dimensional Fourier transform

- The convolution theorem.
  - The convolution theorem states that convolution in time domain corresponds to multiplication in frequency domain and vice versa:
  - It is enormous saving computing time compared to the direct method.

$$h(t) * x(t) \overset{F}{\leftrightarrow} H(\omega)X(\omega)$$

Some properties of the two dimensional Fourier transform

- The DC coefficient
  - The value *F(o,o)* of the DFT is called the DC coefficient. If we put u=v=o in the definition given in equation

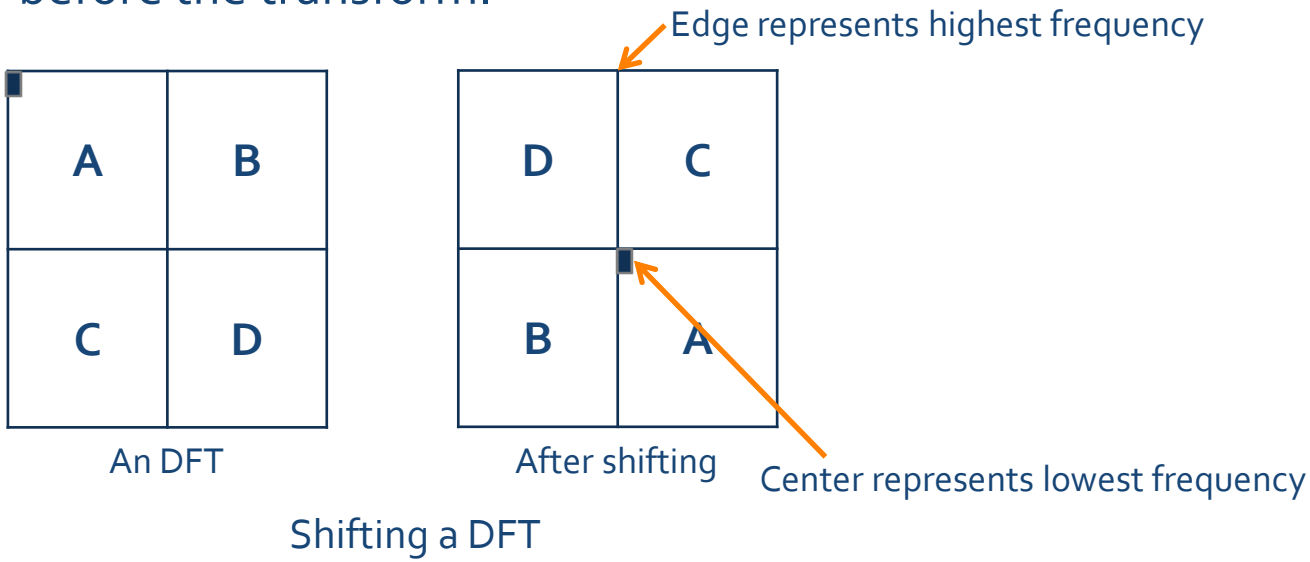
$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\exp(0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

  - That is, this term is equal to the sum of all terms in the original matrix.

```
x=np.array([1, 2, 3, 4])
fft.fft(x)
array([10.+0.j, -2.+2.j, -2.+0.j, -2.-2.j])
```

Some properties of the two dimensional Fourier transform

- Shifting
  - For purposes of display, it is convenient to have the DC coefficient in the centre of the matrix.
  - This will happen if all elements f(x,y) in the matrix are multiplied by $(-1)^{x+y}$ before the transform.



Shifting a DFT

Some properties of the two dimensional Fourier transform

- Conjugate symmetry
  - The half of the transform is a mirror image of the conjugate of the other half. We can think of the top and bottom halves, or the left and right halves, being mirror images of the conjugates of each other.
  - The symmetry means that its information is given in just half of a transform, and the other half is redundant.

```
>> x=[1 2 3 4 5]'
>> y=fft2(x)

y =

   15.0000 + 0.0000i
   -2.5000 + 3.4410i
   -2.5000 + 0.8123i
   -2.5000 - 0.8123i
   -2.5000 - 3.4410i
```

Fourier transforms in Python

- These functions are available in Python with the command from numpy.fft import *
  - fft which takes the DFT of a vector
  - ifft which takes the inverse DFT of a vector
  - fft2 which takes the DFT of a matrix
  - ifft2 which takes the inverse DFT of a matrix
  - fftshift which shifts a transform as shown in figure

Example 1

- Suppose we take a constant matrix f(x,y)=1. There is no corrugations to form a constant matrix. Do the DFT consists of a DC coefficient and zeroes everywhere else.

```
a1=np.ones((8,8))
x1=np.int16(np.fft.fft2(a1))e
```

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

a1

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

X1

Example 2

- Now we'll take a matrix consisting of a single corrugation:

```
a2=np.array([[100,200],[100,200]])
a2=np.tile(a2,(4,4))
 x2=np.int16(np.fft.fft2(a2))
```

100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200
100	200	100	200	100	200	100	200

a2

9600	0	0	0	-3200	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

x2

Example 3

- We will take here a single step edge:

```
a3=np.hstack([np.zeros((8,4)),np.ones((8,4))])
x3=np.fft.fftshift(np.int16(np.fft.fft2(a3)))
```

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

a3

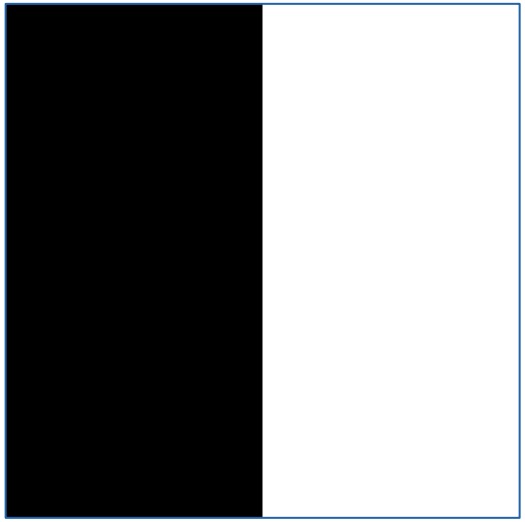
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	-8	0	-8	32	-8	0	-8
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

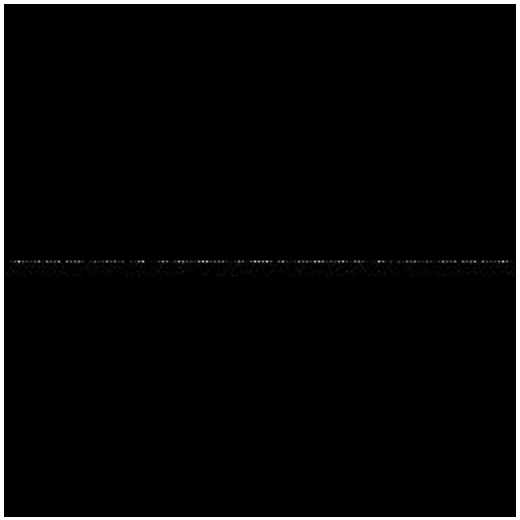
x3

Fourier transforms of images

- We shall produce a simple image consisting of a single edge:

```
a=np.hstack([np.zeros((256,128)),np.ones((256,128))])
x=np.fft.fftshift(np.fft.fft2(a))
xl=np.abs(x)
```



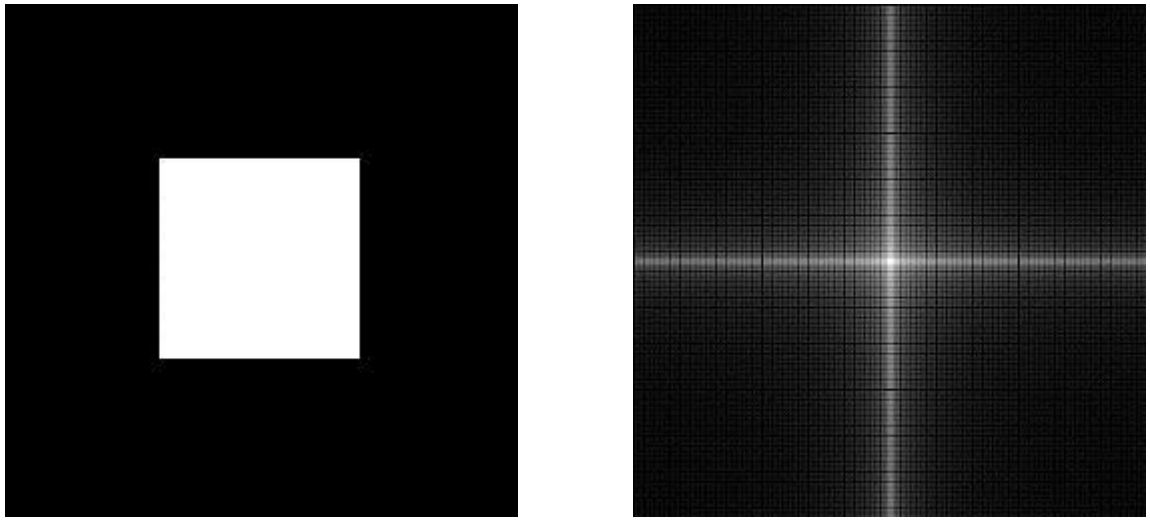


A single edge and its DFT

Fourier transforms of images

- Now we'll create a box, and then its Fourier transform:

```
a=np.zeros((256,256))
a[77:177,77:177]=1
af=fft.fftshift(fft.fft2(a))
af1=ex.rescale_intensity(np.log(1+abs(af)),out_range=(0.0,1.0))
```

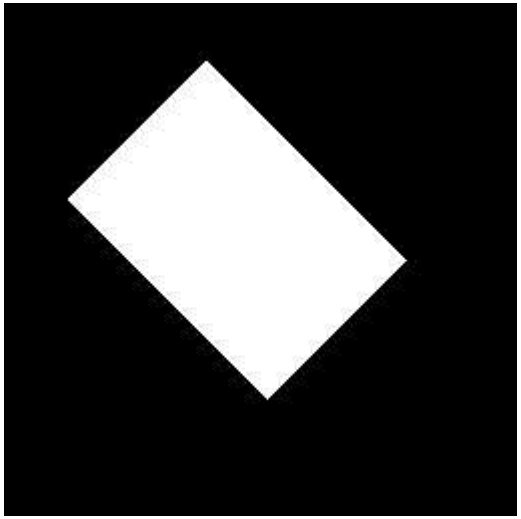


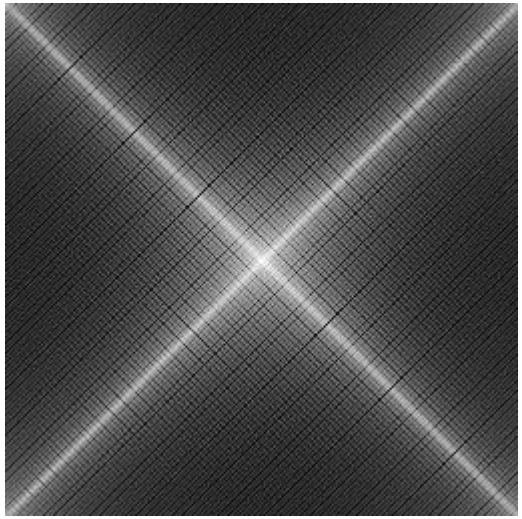
A box and its DFT

Fourier transforms of images

- Now we shall look at a box rotated 45°.

```
x,y=np.meshgrid(range(256),range(256))
b=(x+y<329)&(x+y>128)&(x-y>-67)&(x-y<73)
bf=fft.fftshift(fft.fft2(b))
bfl=np.log(1+np.abs(bf))
bf2=ex.rescale_intensity(np.log(1+abs(bfl)),out_range=(0.0,1.0))
```



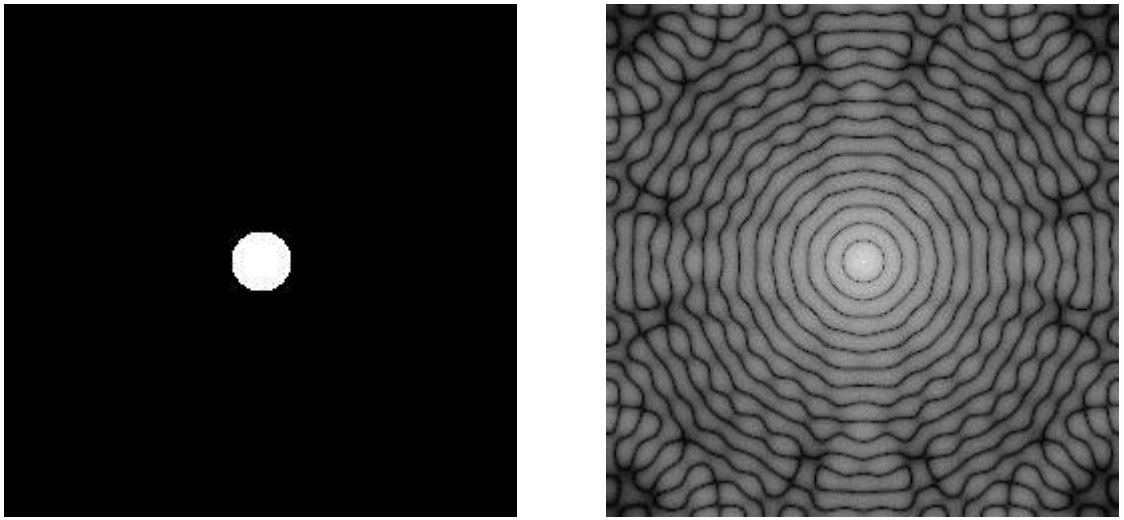


A rotated box and it DFT

Fourier transforms of images

- We will create a small circle, and then transform it:

```
ar=np.arange(-128,128)
x,y=np.meshgrid(ar,ar)
z=np.sqrt(x**2+y**2)
c=(z<15)+1
cf=fft.fftshift(fft.fft2(c))
cfl=np.log(1+np.abs(cf))
cf2=ex.rescale_intensity(np.log(1+abs(cfl)),out_range=(0.0,1.0))
```



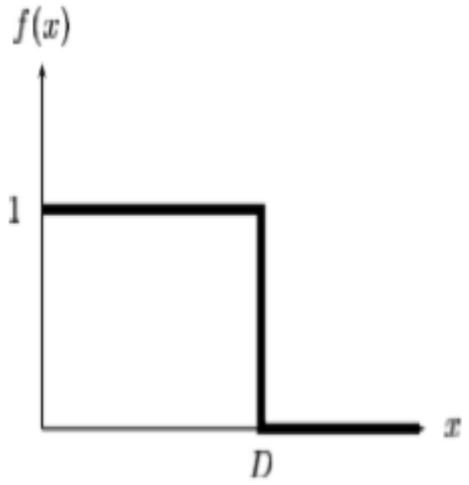
A circle and its DFT

Filtering in frequency domain

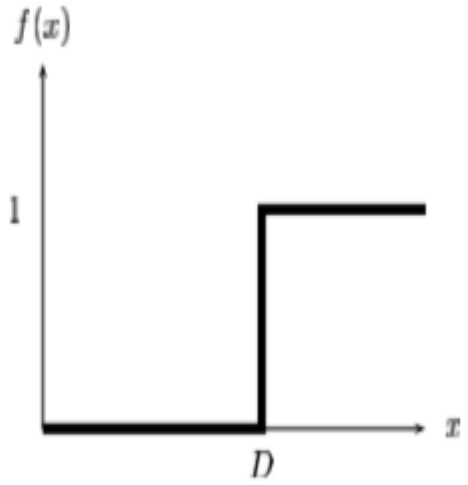
- In frequency domain, a spatial convolution can be performed by element- wise multiplication of the Fourier transform by a suitable "filter matrix".
- Ideal filtering
  - Low pass filtering
  - High pass filtering
- Butterworth filtering
- Gaussian filtering

Ideal filtering

- An ideal low pass filter eliminates all frequencies above the cutoff frequency while passing those below unchanged:
- A ideal high pass filter attenuate the lower frequencies of the input signal while the higher frequencies are passed.



Low pass



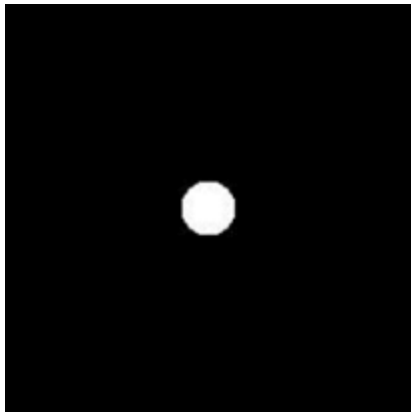
High pass

Ideal filter functions

Low pass filtering

- Suppose we have a Fourier transform matrix F, shifted so that the DC coefficient is in the centre.
- Since the low frequency components are towards the centre, we can perform low pass filtering by multiplying the transform by a matrix in such a way that centre values are maintained, and values away from the centre are either removed or minimized.
- For an ideal low pass filter, this function can be expressed as where D is the cutoff radius.

$$f(x) = \begin{cases} 1 & if x < D, \\ 0 & if x \geq D, \end{cases}$$

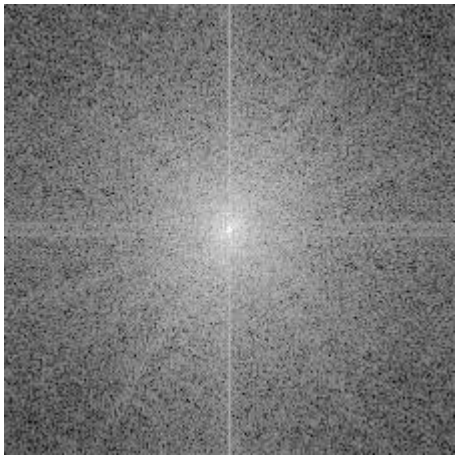


Ideal low pass filter with cutoff radius D = 10

Low pass filtering

- Let's see what happens if we apply this filter to an image.

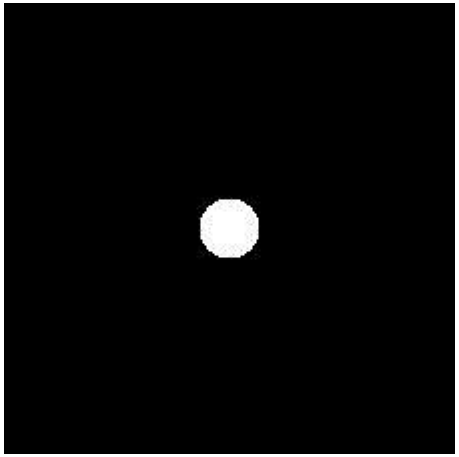
```
img=io.imread('images/waterlily.jpg')
imghsv=co.rgb2hsv(img)
imgf=fft.fftshift(fft.fft2(imghsv[:,:,0]))
imgfl=np.log(1+np.abs(imgf))
imgf2=ex.rescale_intensity(np.log(1+abs(imgfl)),out_range=(0.0,1.0))
```



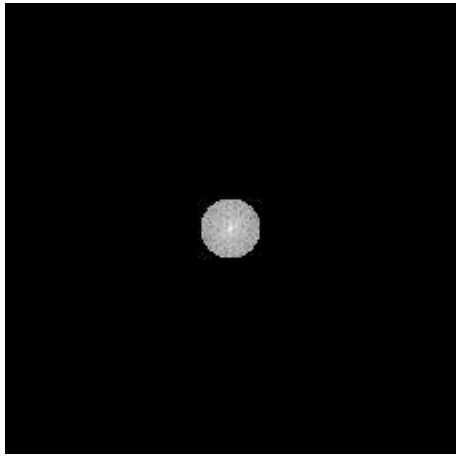
The waterlily image and its DFT

Low pass filtering (cont.)

- Now we can perform a low pass filter by multiplying the transform matrix by the circle matrix



Ideal low pass filter



Ideal filtering on the DFT

After Inversion

Applying ideal low pass filtering

Low pass filtering (cont.)

Cutoff radius D = 10

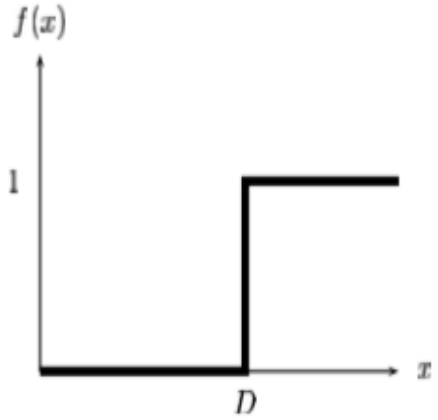
Cutoff radius D = 30

Ideal low pass filtering with different cutoffs

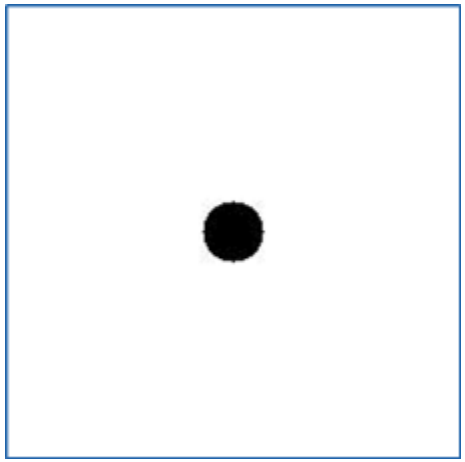
High pass filtering

- high pass filtering can be performed by the opposite: eliminating centre values and keeping the others.
- Then the ideal high pass filters can be described similarly:

$$f(x) = \begin{cases} 1 & if x < D, \\ 0 & if x \geq D, \end{cases}$$



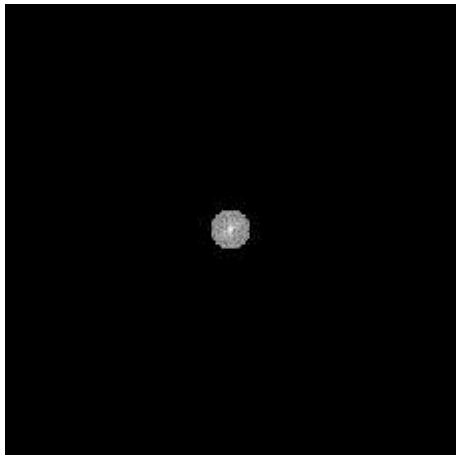
High pass function



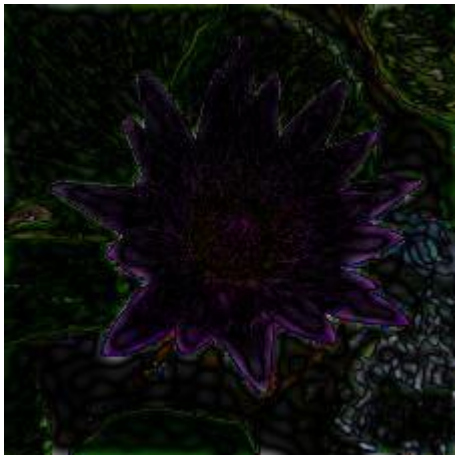
```
x,y=np.meshgrid(ar,ar)
c=(x**2+y**2>d**2)*1
imgfl=imgf*c
img2=abs(fft.ifft2(imgfl)
```

Ideal low pass filter with cutoff radius D = 10

High pass filtering (cont.)



The DFT after high pass filtering



The result image

Applying an ideal high pass filter to an image

Butterworth filtering

- Ideal filtering simply cuts off the Fourier transform at some distance from the centre. This is very easy to implement, but has the disadvantage of introducing unwanted artifacts: ringing, into the result.
- One way of avoiding this is to use as a filter matrix a circle with a less sharp cutoff. A popular choice is to use Butterworth filters.
- Butterworth filter functions are based on the following functions for low pass filters:

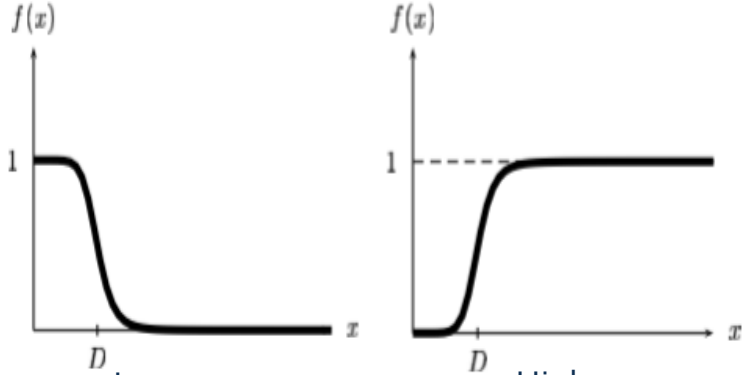
$$f(x) = \frac{1}{1 + (\frac{x}{D})^{2n}} \qquad f(x,y) = \frac{1}{1 + \left(\frac{x^2 + y^2}{D^2}\right)^n}$$

- and for high pass filters:

$$f(x) = \frac{1}{1 + (\frac{D}{x})^{2n}} \qquad f(x,y) = \frac{1}{1 + \left(\frac{D^2}{x^2 + y^2}\right)^n}$$

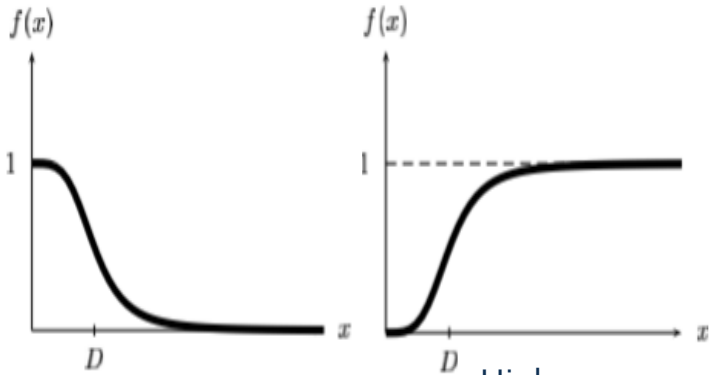
- where in each case the parameter n is called the order of the filter. The size of n dictates the sharpness of the cutoff.

Butterworth filtering



Low pass High pass

Butterworth filter functions with n = 4



Low pass High pass

Butterworth filter functions with n = 2

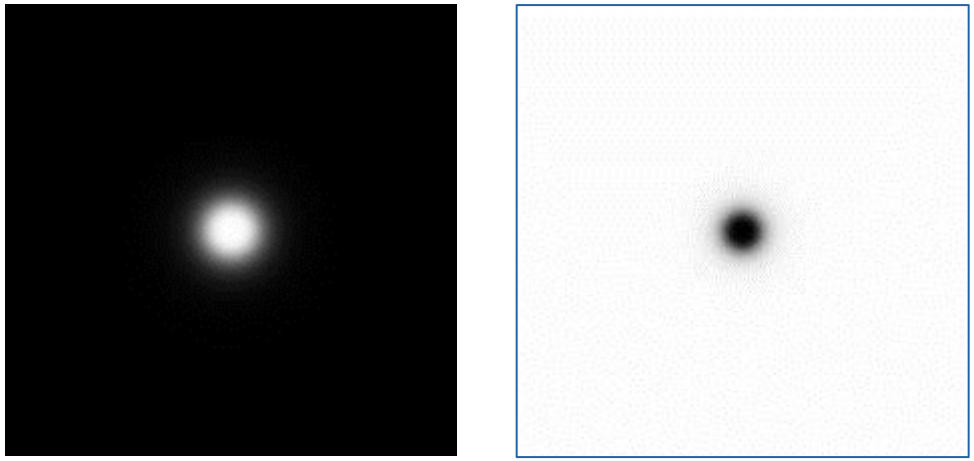
Butterworth filtering

- The commands to produce a Butterworth low pass filter of size 225 x 225 with D = 10 and order n =2:

```
ar=range(-112,113)
x,y=np.meshgrid(ar,ar)
bl=1/(1+((x**2+y**2)/D**2)**2)
```

- Since a Butterworth high pass filter can be obtained by subtracting a low pass filter from 1, we can write general functions to generate Butterworth filters :

```
bh=1-1/(1+((x**2+y**2)/D**2)**2)
```

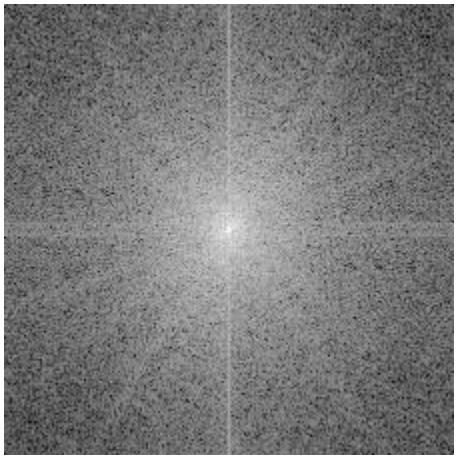


Butterworth low pass and high pass filters with D=10

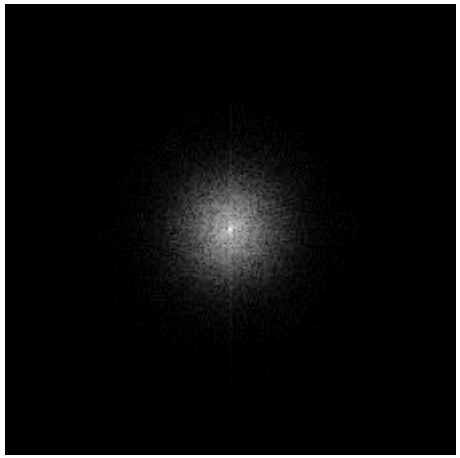
Butterworth filtering (low pass)

- So to apply a Butterworth low pass filter to the DFT of the cameraman image:

```
ar=range(-112,113)
x,y=np.meshgrid(ar,ar)
bl=1/(1+((x**2+y**2)/D**2)**2)
imgfl=imgf*bl
img2=np.abs(fft.ifft2(imgfl))
```



The DFT of image

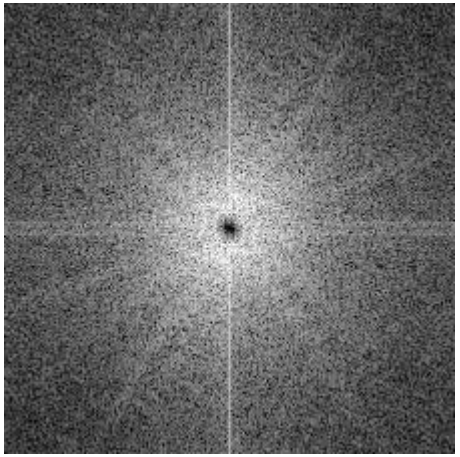


The DFT after filtering

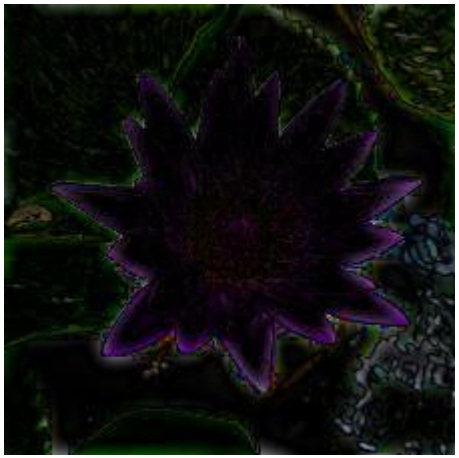
The resulting image

Butterworth filtering(high pass)

- We can apply a Butterworth high pass filter similarly, first by creating the filter and applying it to the image transform:
- and then inverting and displaying the result:



The DFT after filtering



The resulting image

```
ar=range(-112,113)
x,y=np.meshgrid(ar,ar)
bh=1-1/(1+((x**2+y**2)/D**2)**2)
imgfl=imgf*c
img2=abs(fft.ifft2(imgfl))
```

Butterworth low pass filtering

Gaussian filtering

- Gaussian filters could be used for low pass, high pass filtering
- The implementation is very simple: create a Gaussian filter, multiply it by the image transform, and invert the result.
- Fourier transform of a Gaussian is a Gaussian, we should get exactly the same results as when using a linear Gaussian spatial filter.
- Gaussian filters may be considered to be the most "smooth" of all the filters we have discussed so far, with ideal filters the least smooth, and Butterworth filters in the middle.

Gaussian filtering

- In Python, Gaussian filters are provided in the ndimage of scipy, and there is a wrapper in skimage.filter. However, we can also create a filter for ourselves. Using the image of size (225 x 225) , and a standard deviation σ = 10, define:

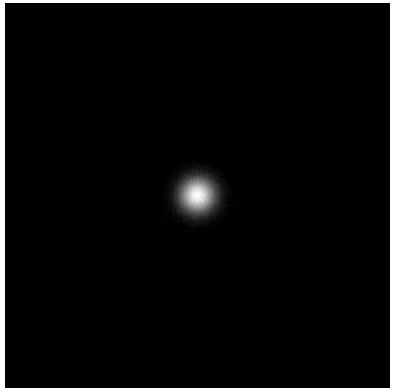
```
sigma=10
g=np.exp(-(x**2+y**2)/sigma**2)
gl=g/g.max()
```

- the standard deviation controls the width of the filter. Clearly, the larger the standard deviation, the wider the function, and so the greater amount of the transform is preserved.

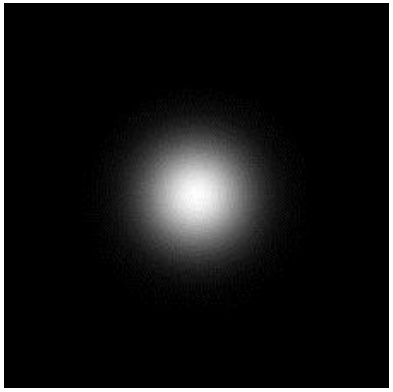
- We can apply a high pass Gaussian filter easily; we create a high pass filter by subtracting a low pass filter from 1.

```
gh=1-(g/g.max())
```

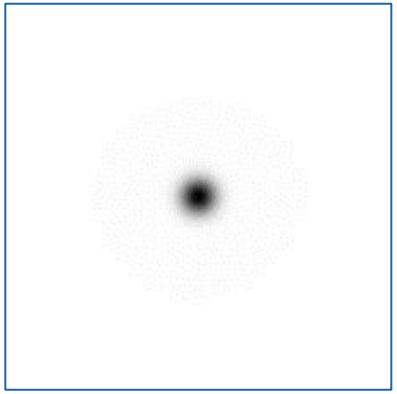
Gaussian filtering



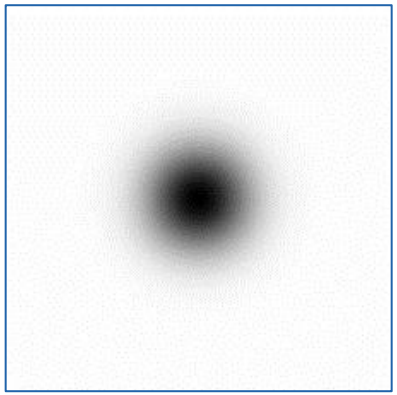
$\sigma = 10$



$\sigma = 30$



$\sigma = 10$



$\sigma = 30$

Gaussian low pass and high pass filters

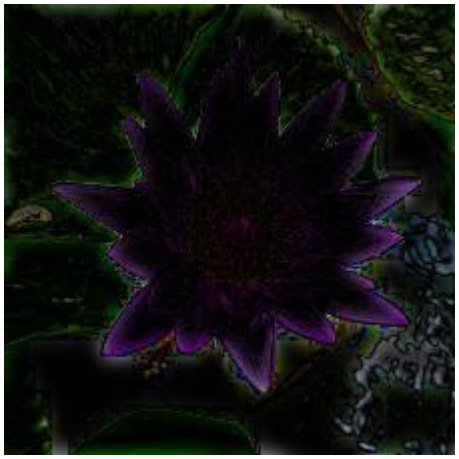
Gaussian filtering (low pass)

$\sigma = 10$

$\sigma = 30$

Applying a Gaussian low pass filter in the frequency domain

Gaussian filtering (high pass)



$\sigma = 10$

$\sigma = 30$

Applying a Gaussian high pass filter in the frequency domain

Assignment

Question1: Write user-defined function in python to perform one dimensional Discrete Fourier Transform and its inverse transform.

Question2: Write user-defined function in python to perform two dimensional Discrete Fourier Transform and its inverse transform.

Python code

```
# one dimensional DFT  forward transform
def DFT_1D(f):
    import numpy as np
    N=f.size
    F=np.zeros(N,dtype=complex)
    for i in range(N):
        for j in range(N):
            F[i]=F[i]+np.exp(np.complex(0,((2*np.pi*i*j)/N)))*f[j]
    for x in F:
        print('%0.2f%+0.2fi'%(x.real,x.imag))
    F=np.around(F,2)
    return F
```

```
#one dimensional DFT  inverse transform
def IDFT_1D(F):
    import numpy as np
     N=F.size
    f=np.zeros(N)
    for i in range(N):
        for j in range(N):
            f[i]=f[i]+np.exp(np.complex(0,((2*np.pi*i*j)/N)))*F[j]/N
    for x in f:
        print('%0.2f'%x)
    return f
```

Python code

```
#Two dimensional DFT forward transform
def DFT_2D(f):
    import numpy as np
    N,M=f.shape
    F=np.zeros((N,M),dtype=complex)
    T=np.zeros((N,M),dtype=complex)
    for i in range(N):
        for j in range(M):
            for k in range(N):
                for l in range(M):
                    T[i,j]=T[i,j]+np.exp(np.complex(0,((2*np.pi)*((i*k/M)+(j*l/N)))))*f[k,l]
            F[i,j]+=T[i,j]
    F=np.around(F,2)
    return
```

```
#Two dimensional DFT inverse transform
def IDFT_2D(F):
    import numpy as np
    N,M=F.shape
    f=np.zeros((N,M))
    T=np.zeros((N,M))
    for i in range(N):
        for j in range(M):
            for k in range(N):
                for l in range(M):
                    T[i,j]=T[i,j]+np.exp(np.complex(0,((2*np.pi)*((i*k/M)+(j*l/N)))))*F[k,l]/(N*M)
            f[i,j]+=T[i,j]
    return f
```

Next Week Lecture (Week6)

- Lecture6:Image Segmentation

Thank You