



**UAI**

UNIVERSIDAD ADOLFO IBÁÑEZ  
FACULTAD DE INGENIERÍA Y CIENCIAS



**iUAI**  
UNIVERSIDAD ADOLFO IBÁÑEZ  
FACULTAD DE INGENIERIA Y CIENCIAS

MDS<sup>2019</sup> PROCESAMIENTO  
DIGITAL DE IMÁGENES

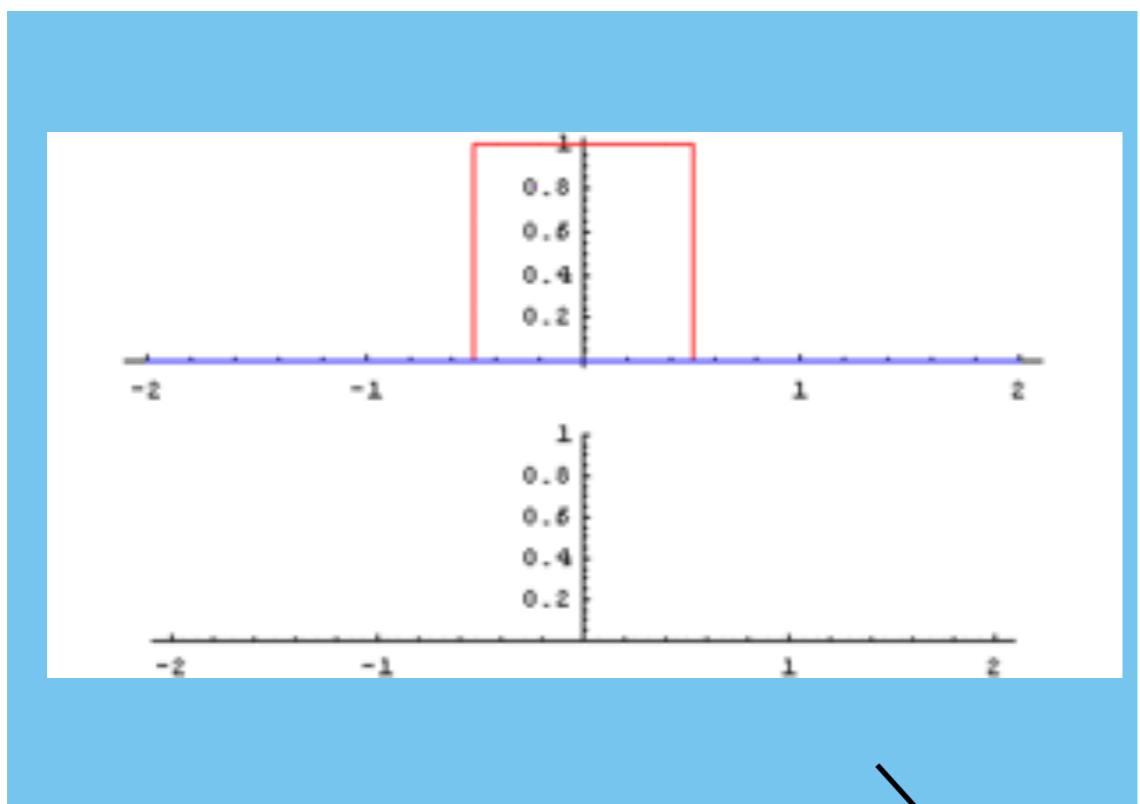
MEJORAMIENTO EN LA FRECUENCIA

Miguel Carrasco  
[miguel.carrasco@uai.cl](mailto:miguel.carrasco@uai.cl)  
1er Semestre 2020

- Mejoramiento en la frecuencia
  - Convolución y Filtrado
  - Filtro Ideal
  - Filtro Gaussiano
  - Filtro Butterworth
  - Filtro Homomórfico

## ► Descripción

- La convolución es un operador matemático que calcula la magnitud en la cual se superponen dos funciones



El resultado que genera el ejemplo se muestra como un cambio de área

contínua

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\eta) \cdot g(t - \eta) d\eta$$

discreta

$$f(m) * g(m) = \sum_n f(n) \cdot g(m - n)$$

- ▶ Propiedad de interés
    - Dentro de las propiedades de la convolución, una que nos interesa especialmente tiene relación con la transformada de Fourier.

## Teorema de convolución

$$F(f * g) = F(f) \cdot F(g)$$

# Transformada de Fourier

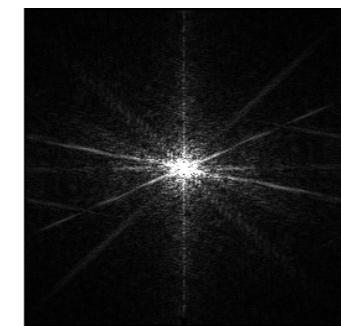
# Multiplicación

## Proceso general

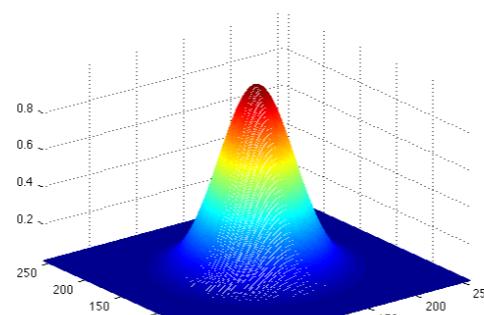


Imagen  
Original

$$f_{i,j}$$



$$F(f)$$



$$H \cdot F$$

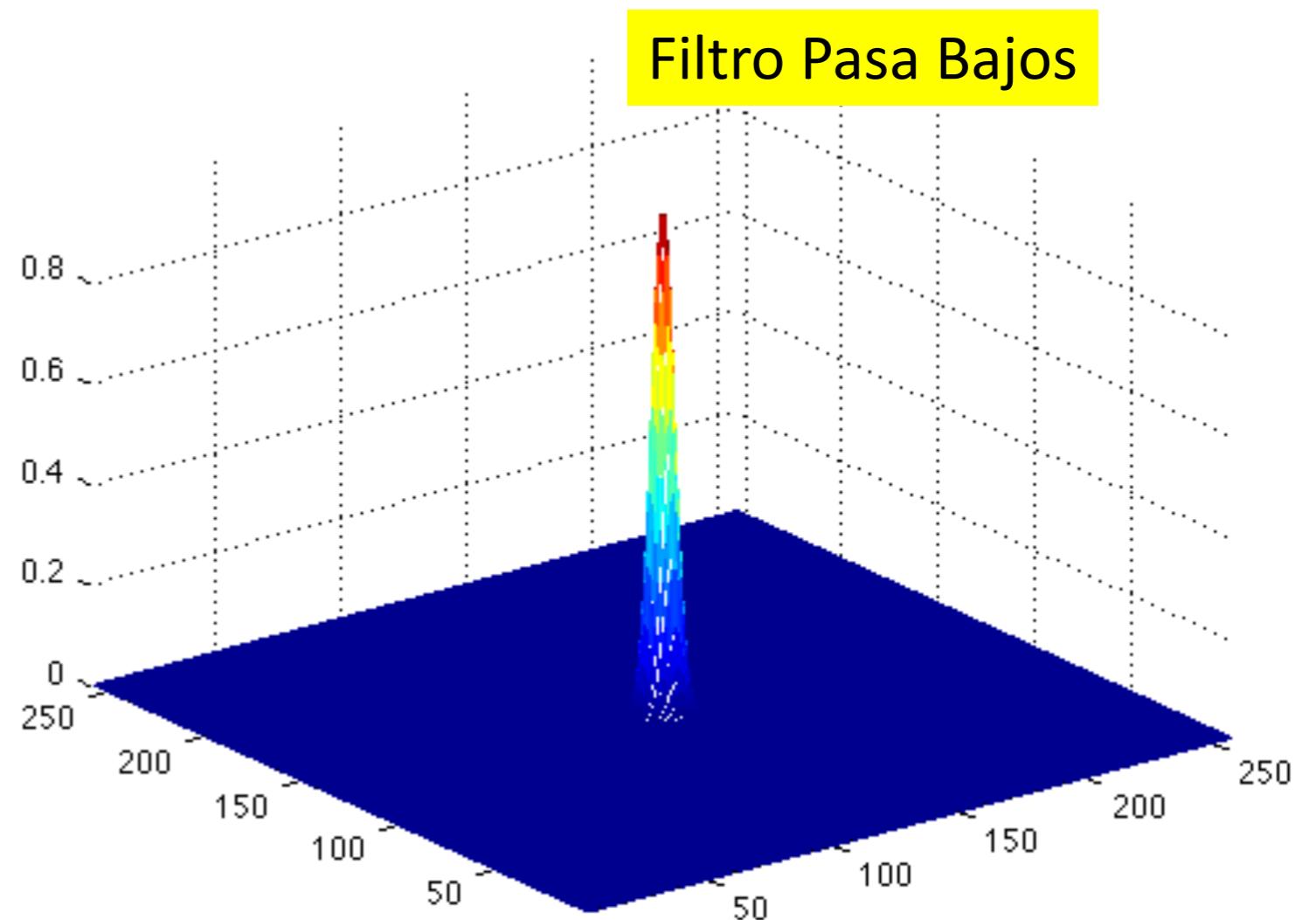


Imagen  
Transformada

$$g_{i,j}$$

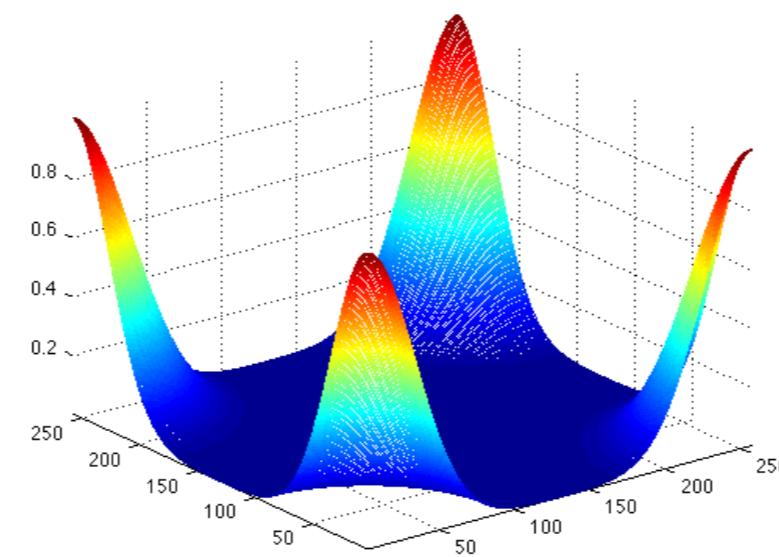
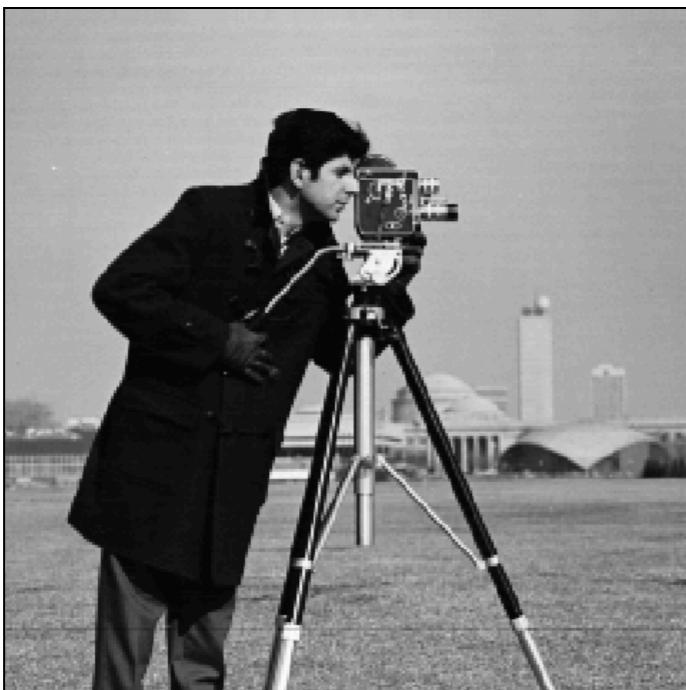
$$F^{-1}(H \cdot F)$$

- ▶ Veamos qué sucede en la imagen si aplicamos este filtro

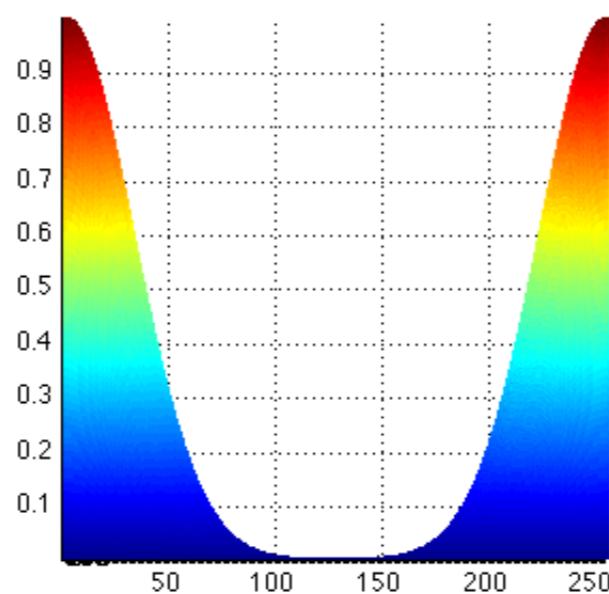


$$H(x, y) = e^{0.005 \cdot (x^2 + y^2)}$$

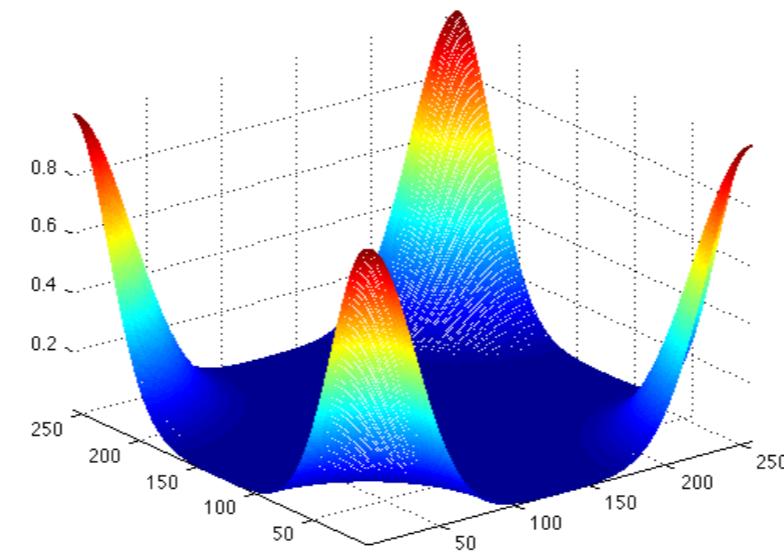
- ▶ Veamos distintos filtros Pasabajos



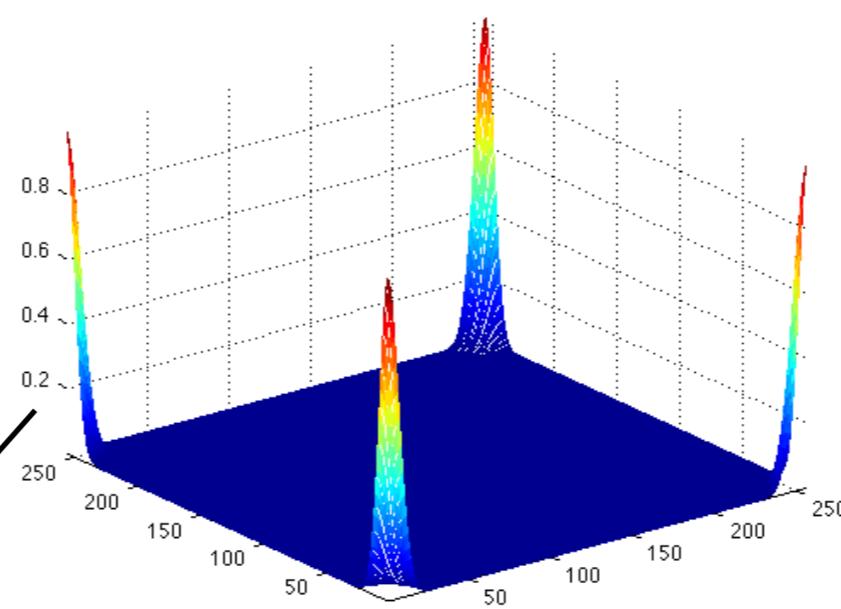
Vista Lateral



- ▶ Veamos distintos filtros Pasabajos



¿Hey, por qué se ve peor?



- Mejoramiento en la frecuencia
  - Convolución y Filtrado
  - Filtro Ideal
  - Filtro Gaussiano
  - Filtro Butterworth
  - Filtro Homomórfico



## ▶ Filtro pasabajos ideal

- El filtro pasabajos permite el paso de las frecuencias más bajas y reduce las altas

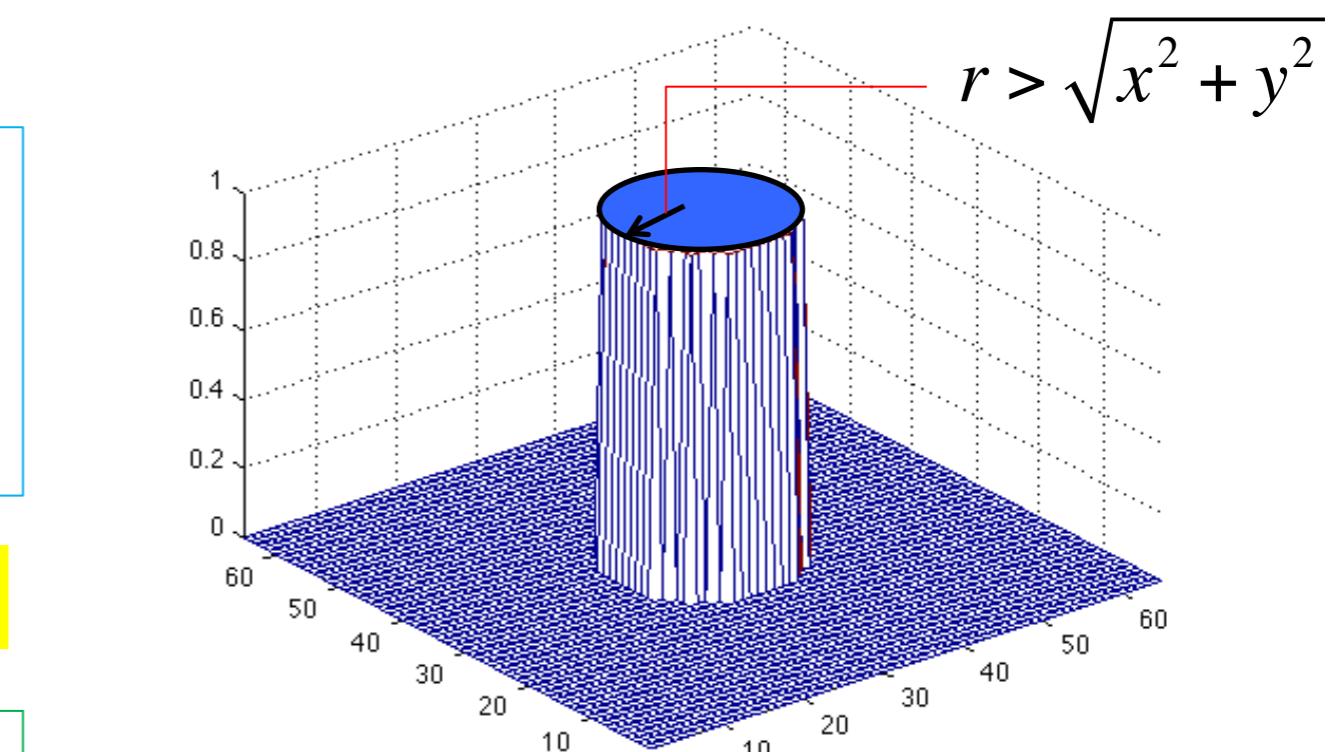


```
import numpy as np

r = 40
x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x,y)
```

¿Qué hace meshgrid?

```
a = [1, 2, 3]
X, Y = np.meshgrid(a,a)
```



X =

1	2	3
1	2	3
1	2	3

Y =

1	1	1
2	2	2
3	3	3



## ▶ Filtro pasabajos ideal

- El filtro pasabajos permite el paso de las frecuencias más bajas y reduce las altas

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

r = 40
x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x, y)

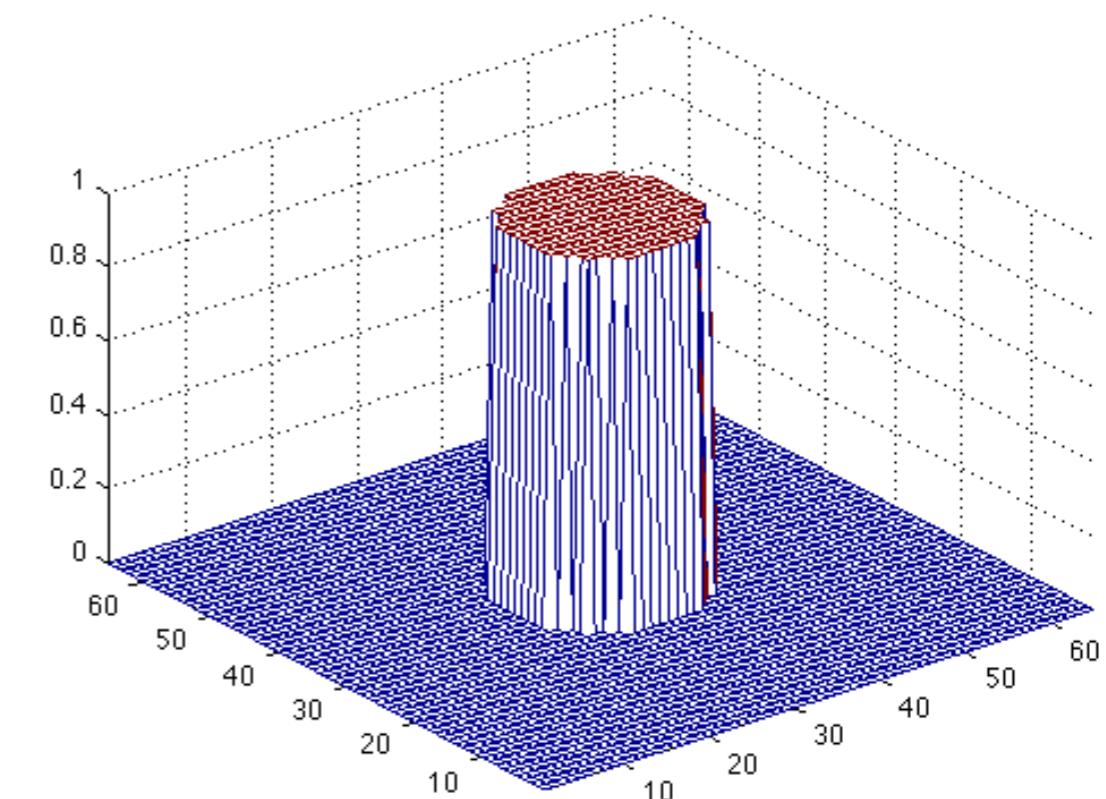
Z = np.sqrt( X**2 + Y**2 )<r
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z)

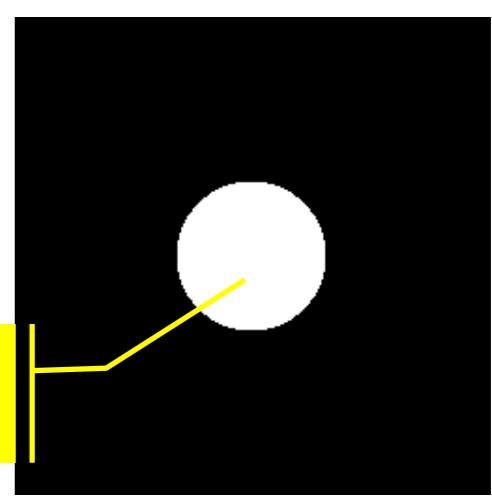
plt.show()
```

Emplear la ecuación paramétrica del círculo

$$r > \sqrt{x^2 + y^2}$$

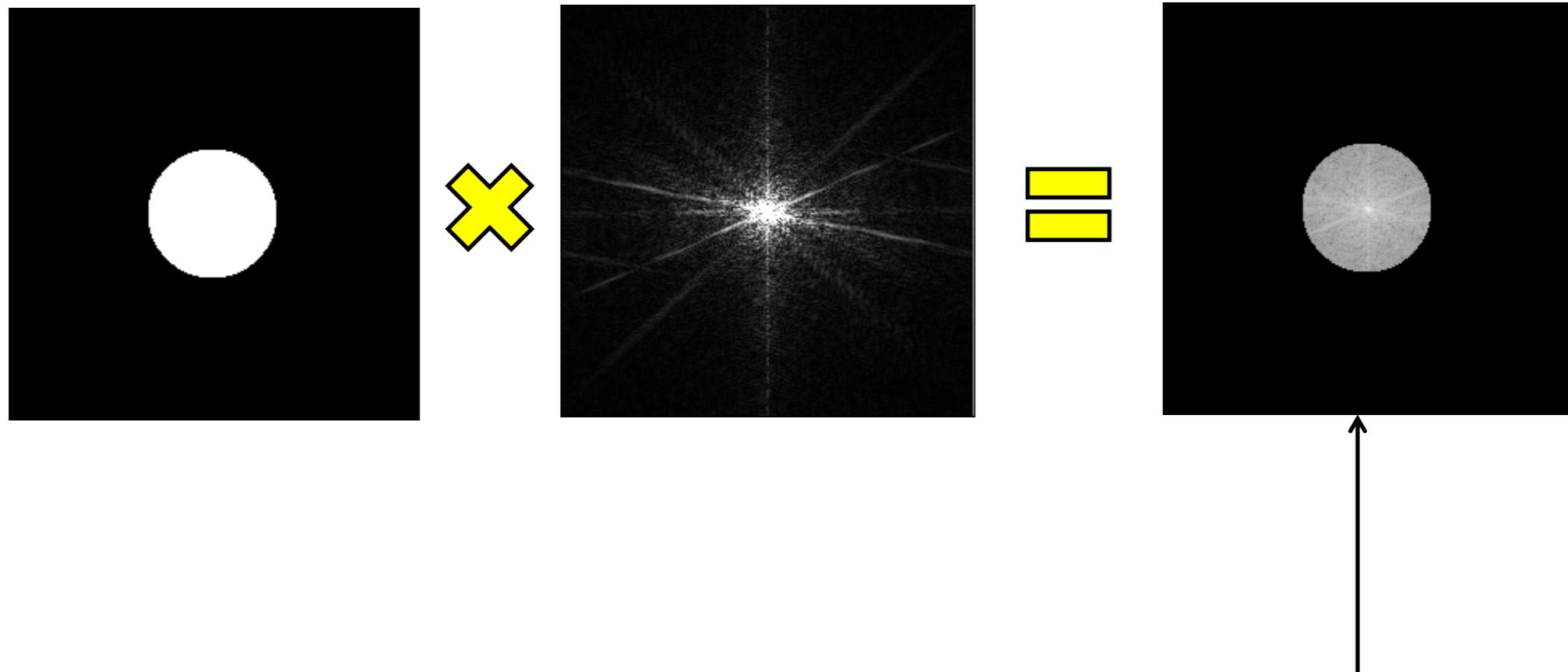


Color negro equivale a cero.



## ▶ Filtro pasabajos ideal

- El filtro pasabajos permite el paso de las frecuencias más bajas y reduce las altas



```
spectrum_filter = np.log(np.abs(fshift)) * Z
out = cv2.normalize(spectrum_filter, None, 0.0, 1.0, cv2.NORM_MINMAX)

cv2.imshow('filter', out)
cv2.waitKey(0)
```

## ▶ Filtro pasabajos ideal

- El filtro pasabajos permite el paso de las frecuencias más bajas y reduce las altas

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

r = 40
x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X,Y = np.meshgrid(x,y)

Z = np.sqrt( X**2 + Y**2 )<r

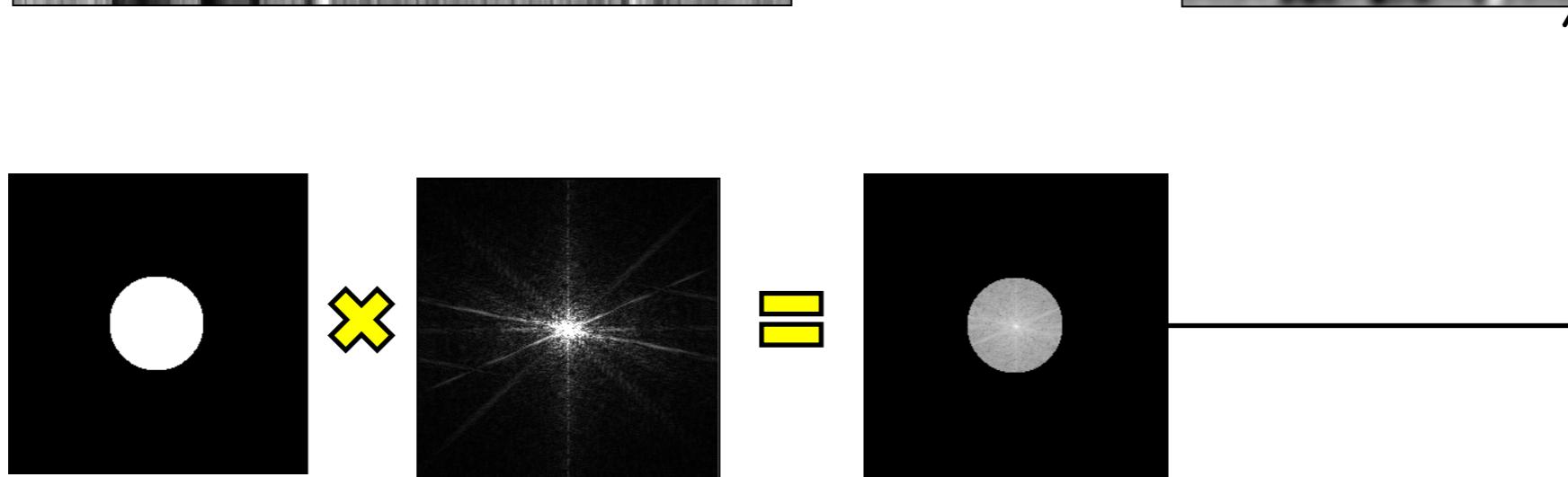
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
F = np.fft.fft2(gray)
fshift = np.fft.fftshift(F)

spectrum_filter = np.log(np.abs(fshift)) * Z
out = cv2.normalize(spectrum_filter, None, 0.0, 1.0, cv2.NORM_MINMAX)

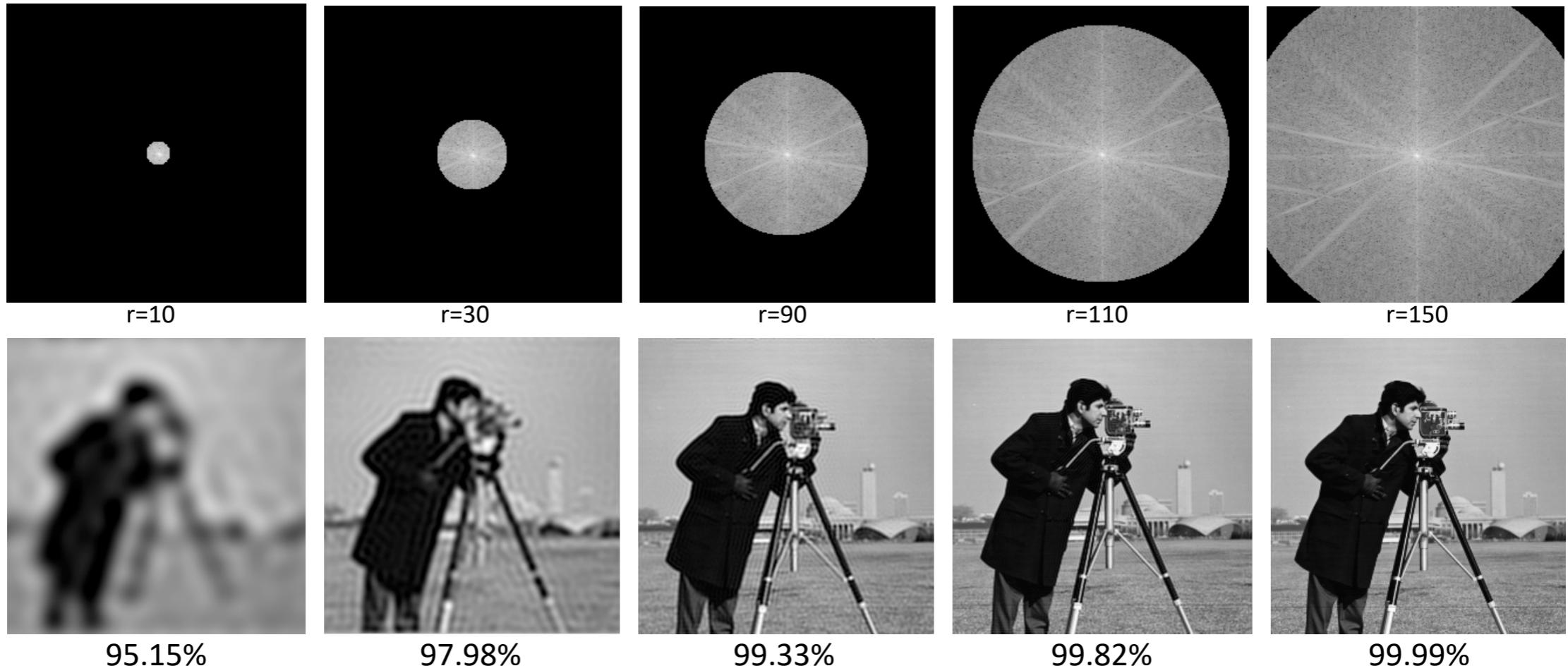
cv2.imshow('filter', out)
cv2.waitKey(0)
```



- ▶ Filtro pasabajos ideal



- Distribución de la Energía



```

F = np.fft.fft2(gray)
FS = np.fft.fftshift(F)
BF = FS * Z

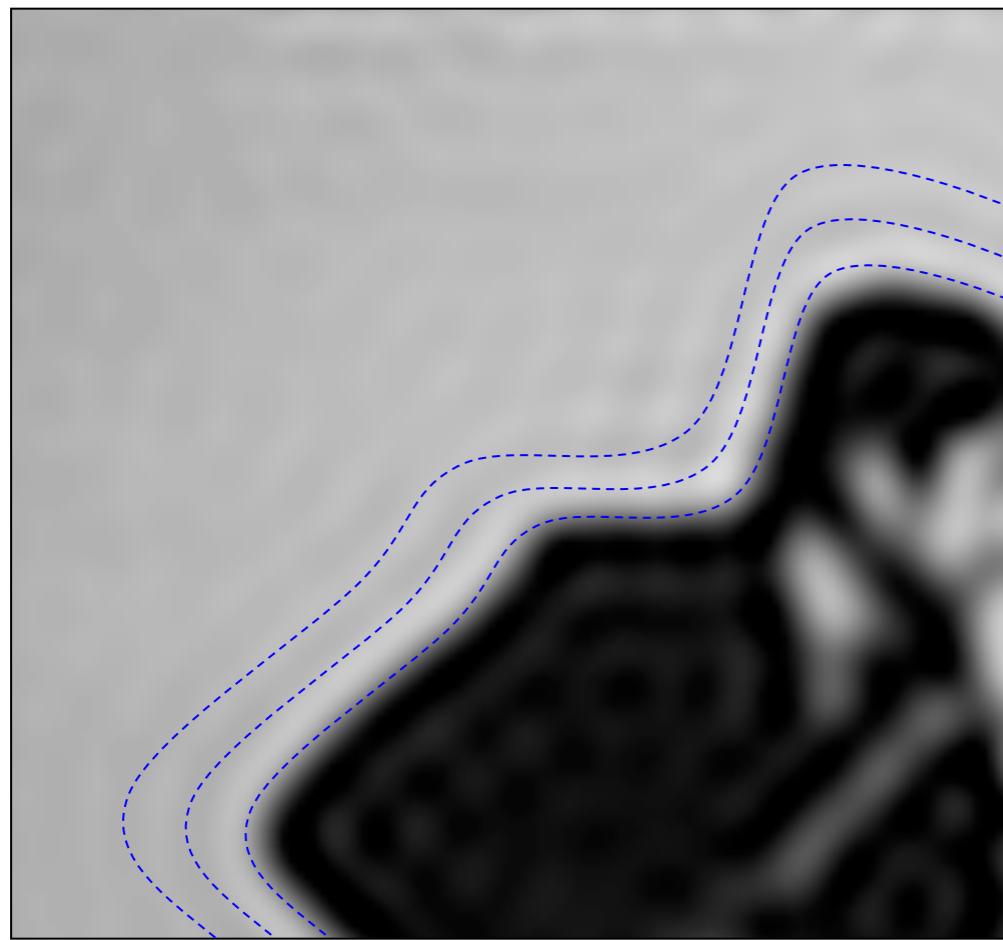
EF = np.sum(abs(np.power(FS, 2)))
EI = np.sum(abs(np.power(BF, 2)))
    
```

Energía:

$$|F(u, v)|^2$$

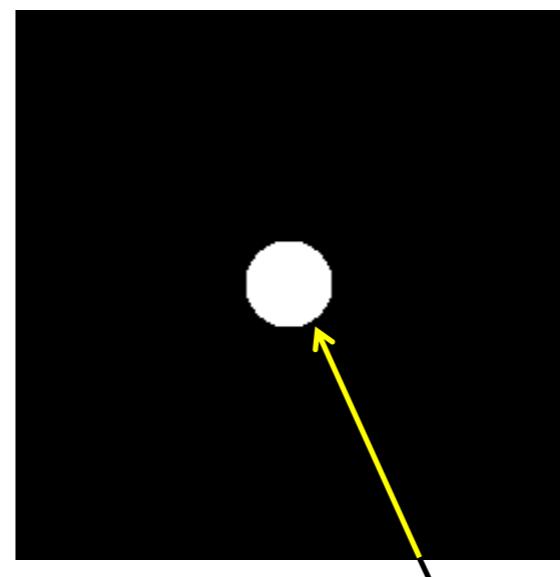


## ▶ Efecto de ringing

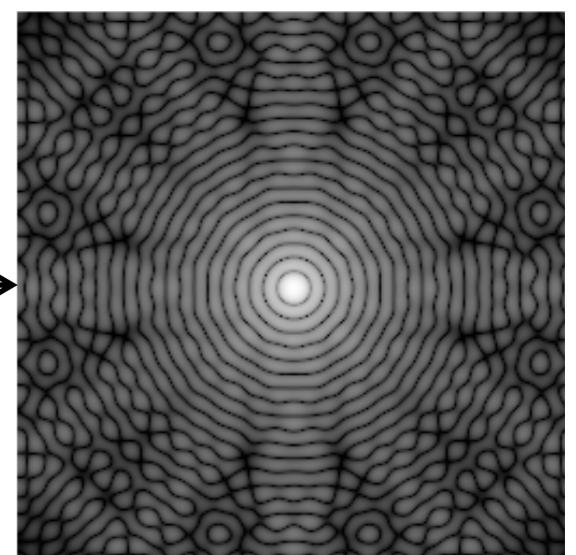


Efecto *ringing*  
fenómeno de gibbs

Filtro Ideal (espacio)



Filtro Ideal (Frecuencia)



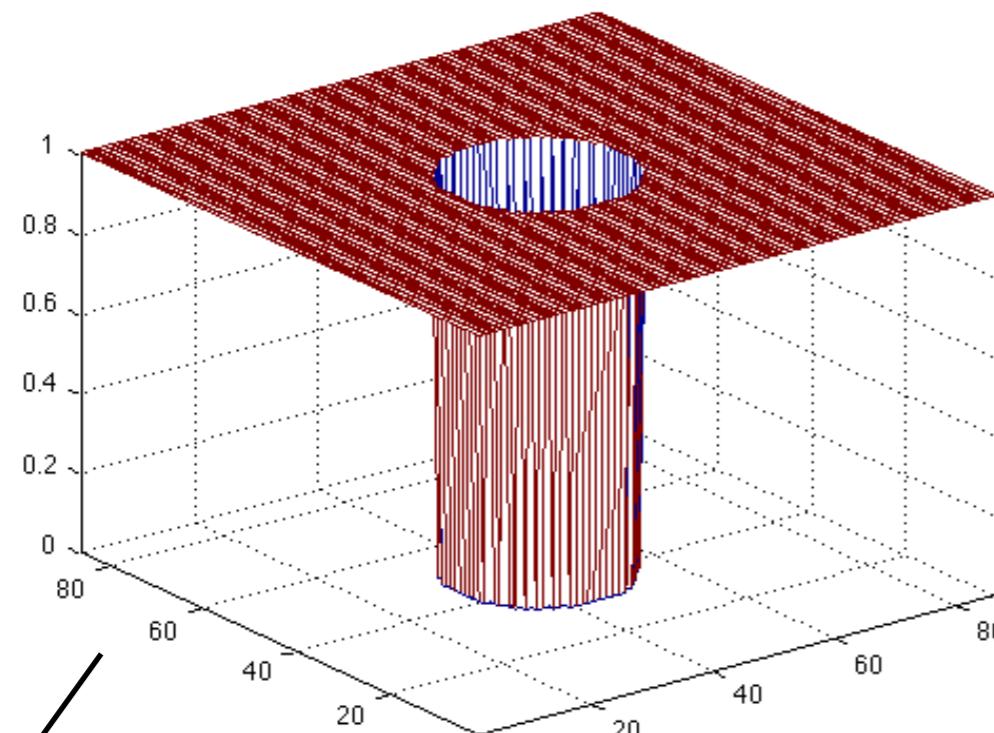
Filtro con  
discontinuidad

El fenómeno de Gibbs se produce por la aproximación de una función discontinua a una serie finita continua de series de senos y cosenos.

A medida que la serie de Fourier tiene más términos, las oscilaciones van siendo más rápidas y más pequeñas.

## ▶ Filtro pasa alto ideal

- El filtro pasa alto permite el paso de las frecuencias más altas y reduce las bajas



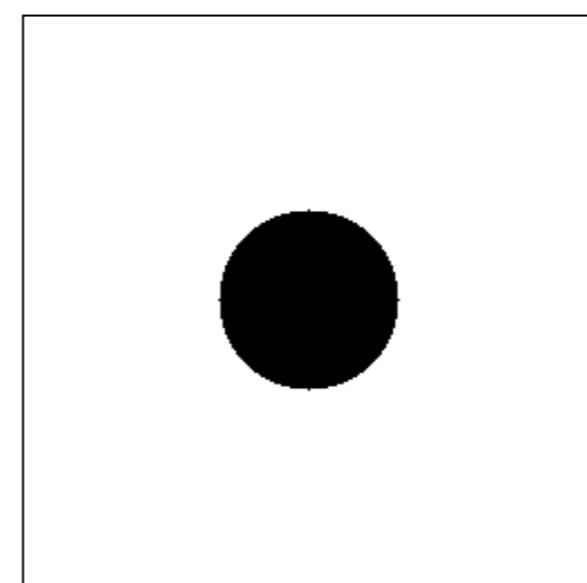
$$r < \sqrt{x^2 + y^2}$$

Empleamos la ecuación paramétrica del círculo

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

r = 40
x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X,Y = np.meshgrid(x,y)

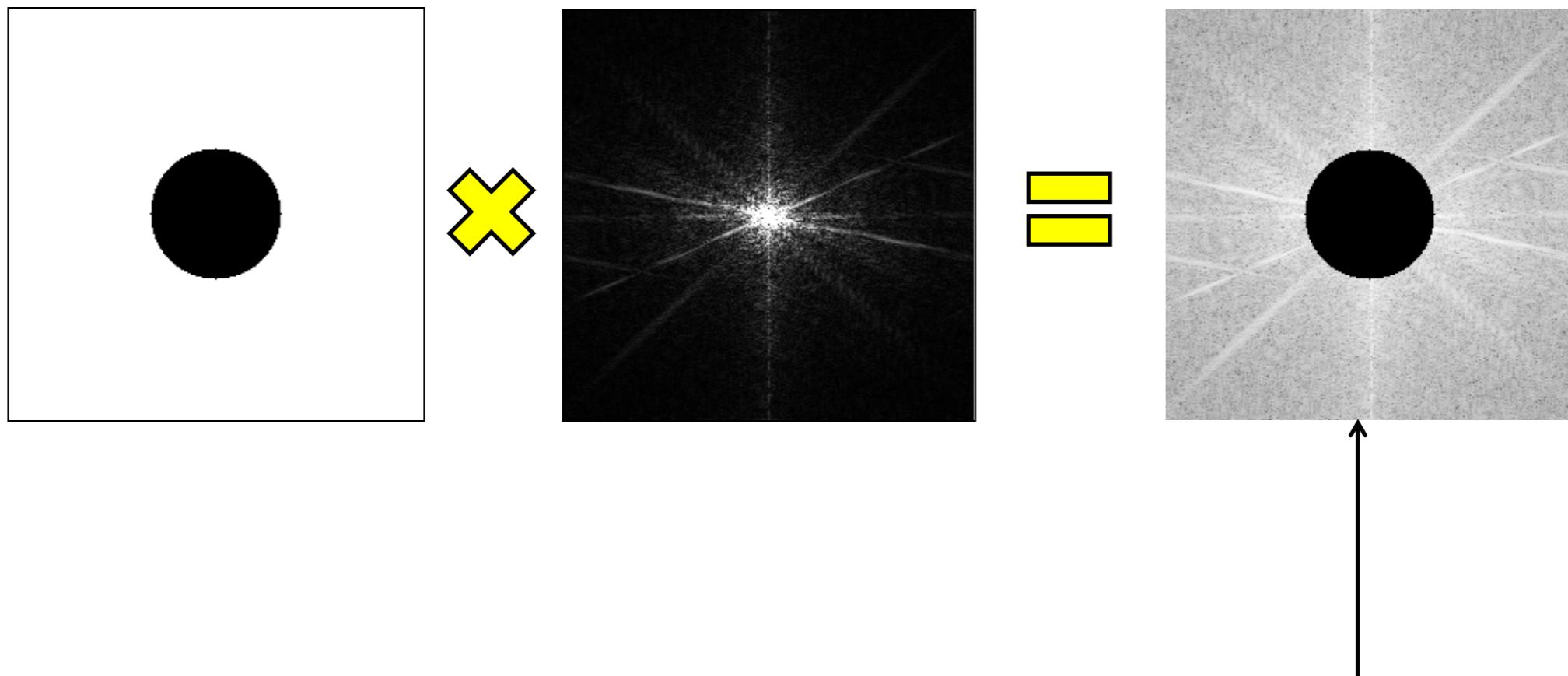
Z = np.sqrt(X**2 + Y**2)>r
```



Color negro equivale a cero.

## ▶ Filtro pasaalto ideal

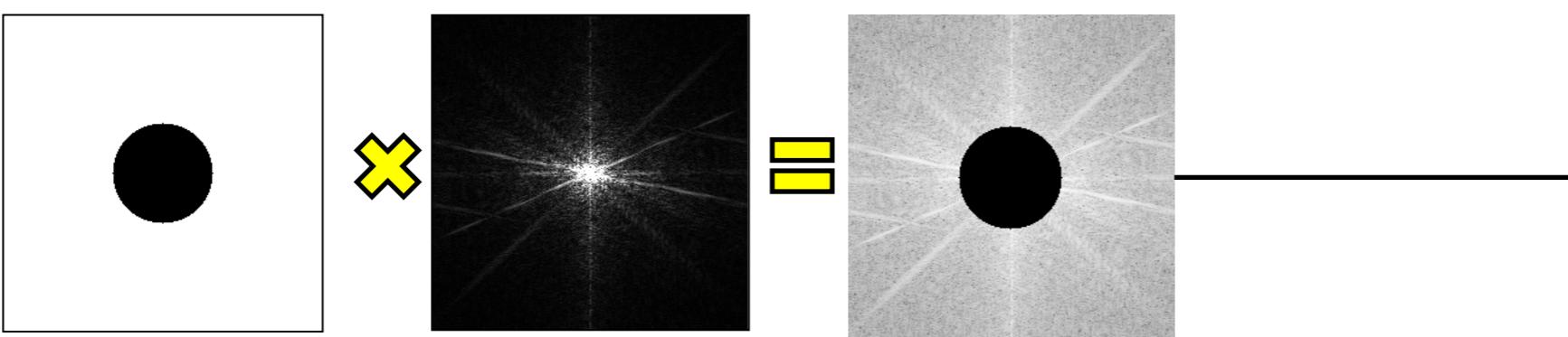
- El filtro pasaalto permite el paso de las frecuencias más altas y reduce las bajas



```
spectrum_filter = np.log(np.abs(fshift)) * Z
out = cv2.normalize(spectrum_filter, None, 0.0, 1.0, cv2.NORM_MINMAX)

cv2.imshow('filter', out)
cv2.waitKey(0)
```

- ▶ Filtro pasaalto ideal



## ▶ Filtro pasabanda ideal

- El filtro pasabanda permite el paso de un rango de frecuencias.

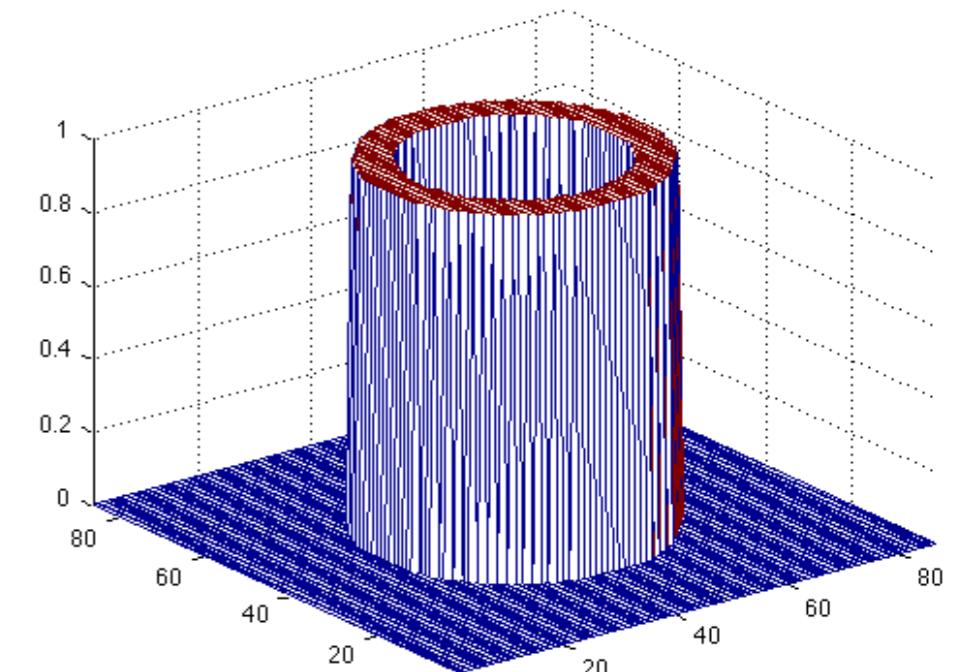
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

r      = 60
c      = 10

x      = np.linspace(-127, 128, 256)
y      = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x,y)

Z1 = np.sqrt( X**2 + Y**2 ) > r-c
Z2 = np.sqrt( X**2 + Y**2 ) < r+c

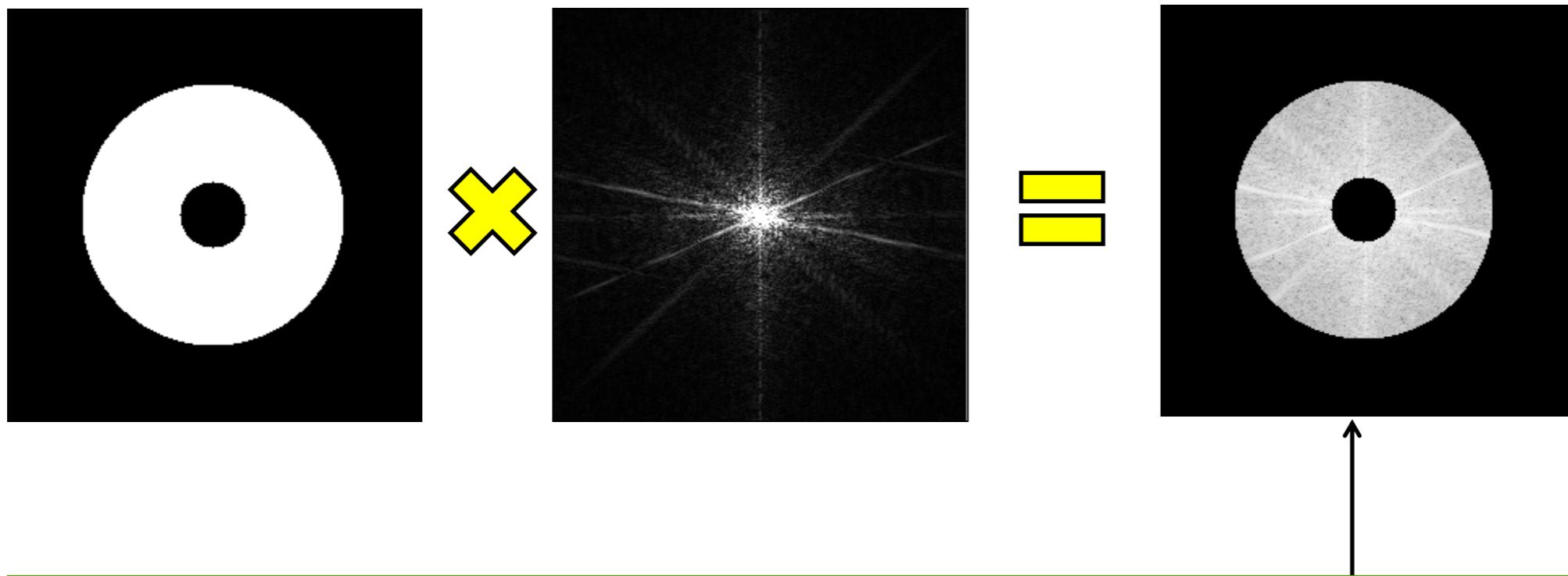
Z = np.bitwise_and(Z1,Z2, out=None)
```



$$\sqrt{x^2 + y^2} > r - d \wedge \sqrt{x^2 + y^2} < r + d$$

Emplear la ecuación  
paramétrica del círculo

- ▶ Filtro pasabanda ideal
  - El filtro pasabanda permite el paso de un rango de frecuencias.

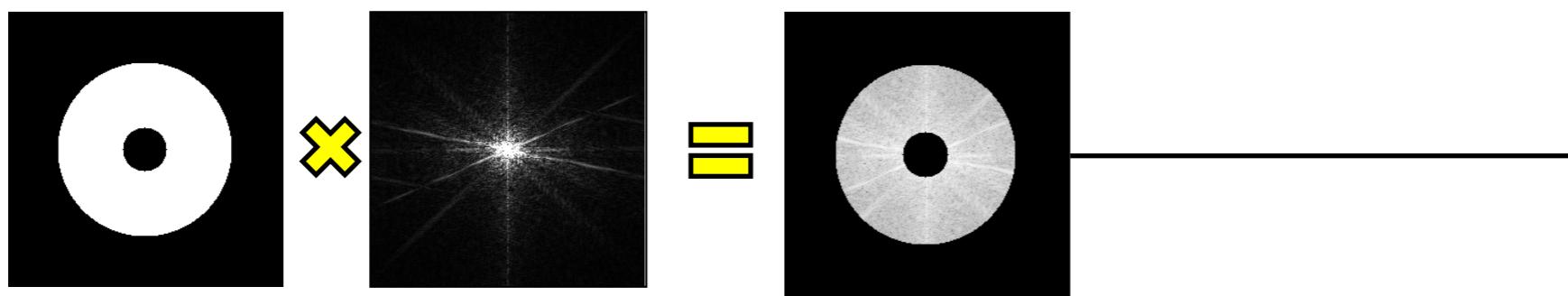


```
F = np.fft.fft2(gray)
FS = np.fft.fftshift(F)
BF = FS * Z

out = cv2.normalize(abs(BF), None, 0.0, 1.0, cv2.NORM_MINMAX)

cv2.imshow('filter', out)
cv2.waitKey(0)
```

- ▶ Filtro pasabanda ideal



- Mejoramiento en la frecuencia
  - Convolución y Filtrado
  - Filtro Ideal
  - Filtro Gaussiano
  - Filtro Butterworth
  - Filtro Homomórfico



## ▶ Filtro pasabajos gaussiano



```
import cv2  
  
img = cv2.imread('cameraman.png')  
  
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```



## ▶ Filtro pasabajos gaussiano



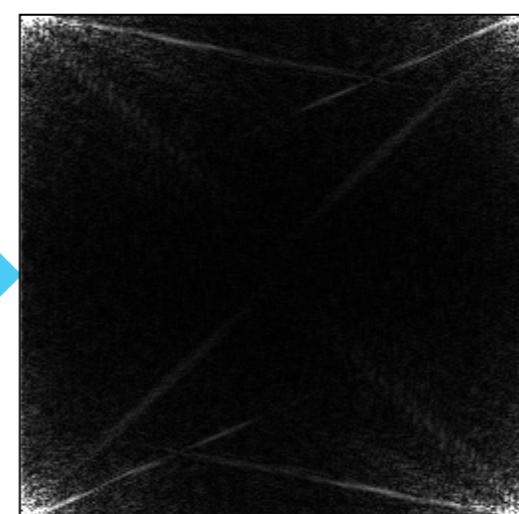
```
import cv2
import numpy as np

img      = cv2.imread('cameraman.png')
gray    = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

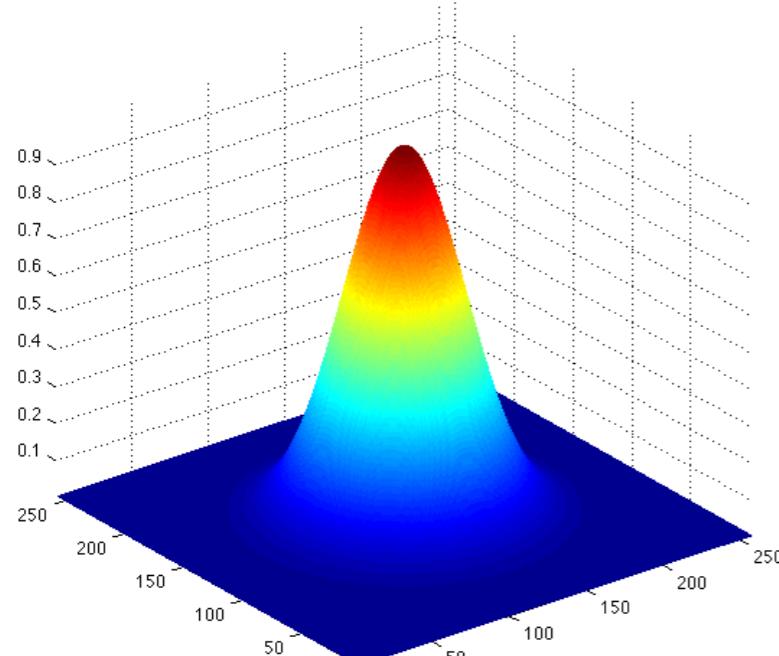
F       = np.fft.fft2(gray)
spectrum = np.log(np.abs(F))

out = cv2.normalize(spectrum, None, 0.0, 1.0, cv2.NORM_MINMAX)
cv2.imshow('fft2', out)
cv2.waitKey(0)
```

Espectro de Fourier



- ▶ Filtro pasabajos gaussiano



```
import numpy as np

x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x,y)
```

Los valores en X,Y corresponden a puntos en el espacio 2D.

```
a = [1, 2, 3]
X, Y = np.meshgrid(a,a)
```

X =

1	2	3
1	2	3
1	2	3

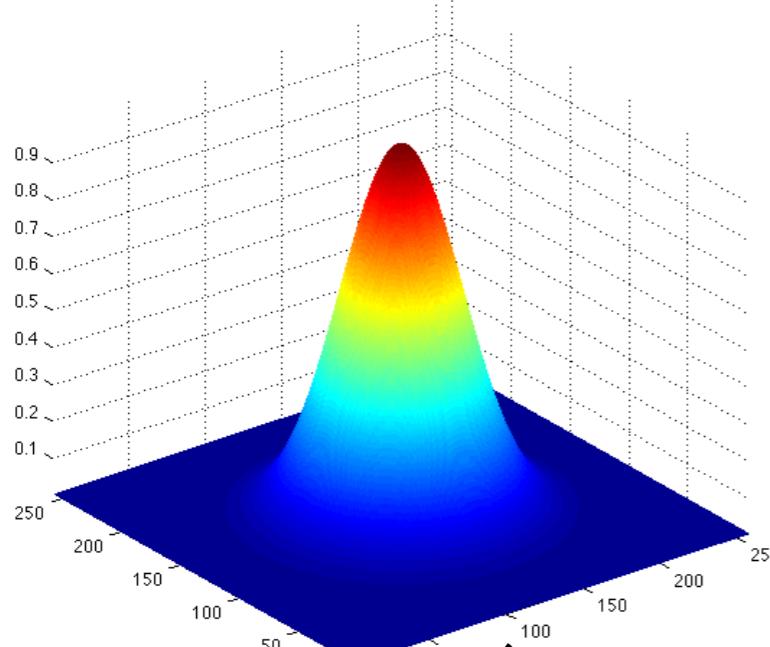
Y =

1	1	1
2	2	2
3	3	3



## ▶ Filtro pasabajos gaussiano

H



$$e^{-0.0005(x^2+y^2)}$$

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

x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x,y)

H = np.exp(-0.0005*(X**2+ Y**2))

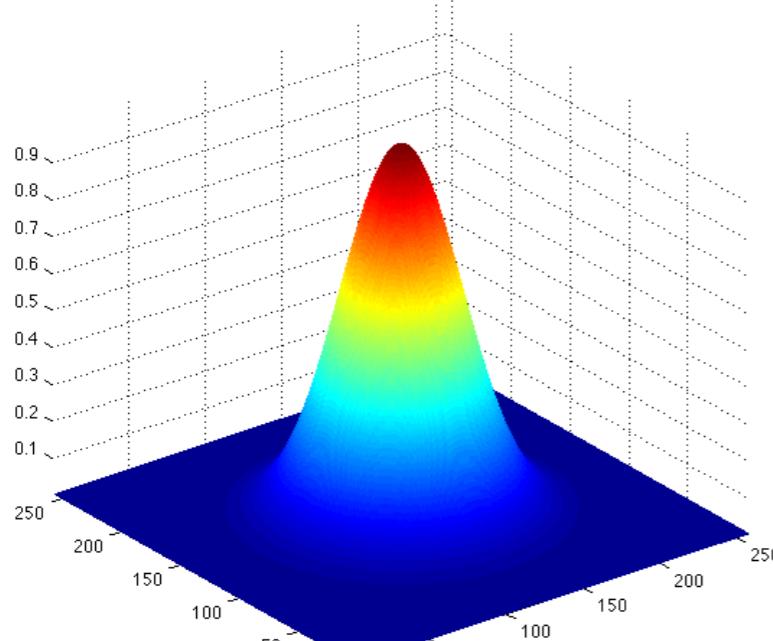
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, H, cmap=cm.Spectral)
plt.show()
```

Con el comando Surface podemos crear una superficie imagen 3D.

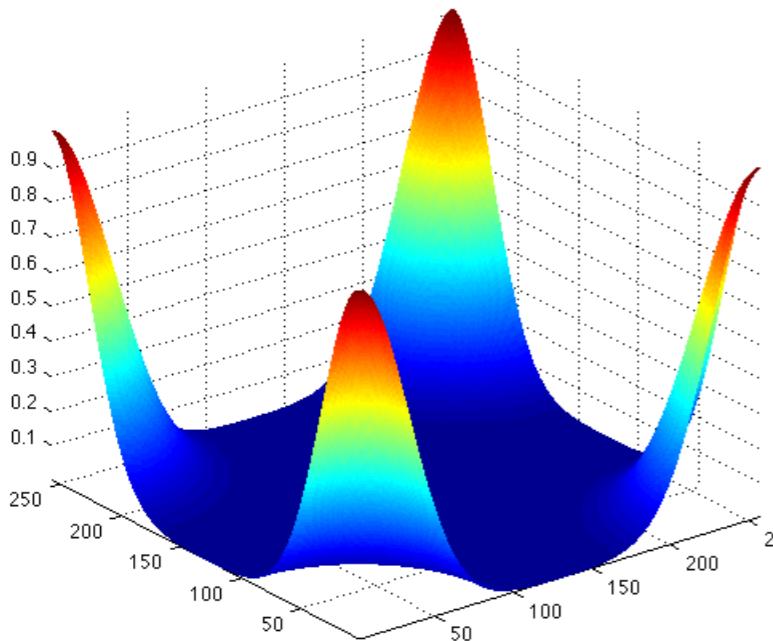


- ▶ Filtro pasabajos gaussiano

H



T



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

x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x,y)

H = np.exp(-0.0005*(X**2+ Y**2))
T = np.fft.fftshift(H)
```

La función fftshift divide la señal en cuadrantes

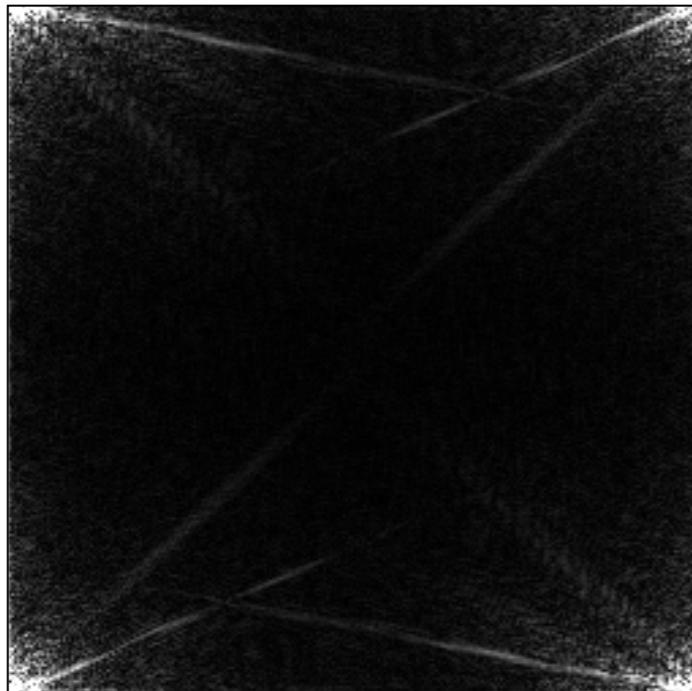
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, T, cmap=cm.Spectral)

plt.show()
```

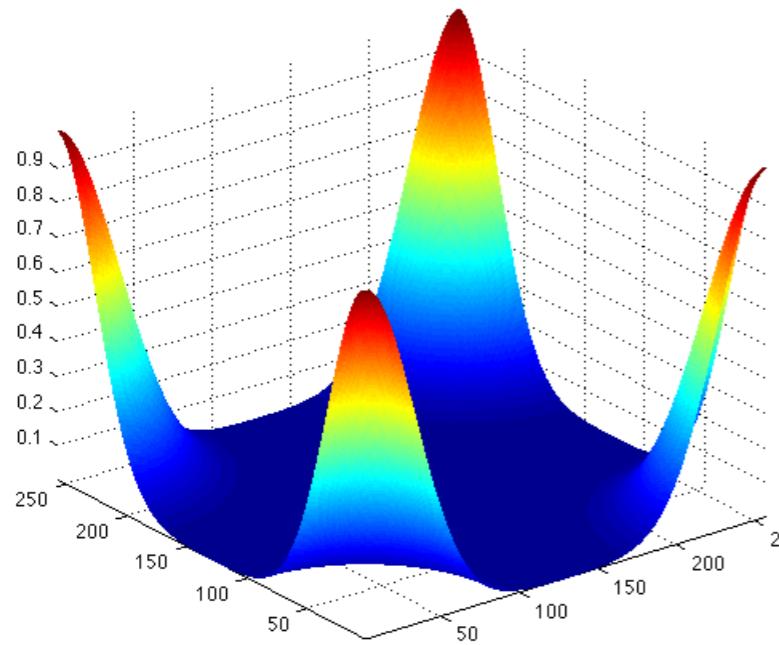


## ▶ Filtro pasabajos gaussiano

F



T



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

#vamos a leer la imagen
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

r = 40
x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x,y)

H = np.exp(-0.0005*(X**2+ Y**2))
T = np.fft.fftshift(H)

F = np.fft.fft2(gray)
BF = T * F
```



## ▶ Filtro pasabajos gaussiano

gray



out



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

#vamos a leer la imagen
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

r = 40
x = np.linspace(-127, 128, 256)
y = np.linspace(-127, 128, 256)
X, Y = np.meshgrid(x, y)

H = np.exp(-0.0005*(X**2+ Y**2))
T = np.fft.fftshift(H)

F = np.fft.fft2(gray)
BF = T * F

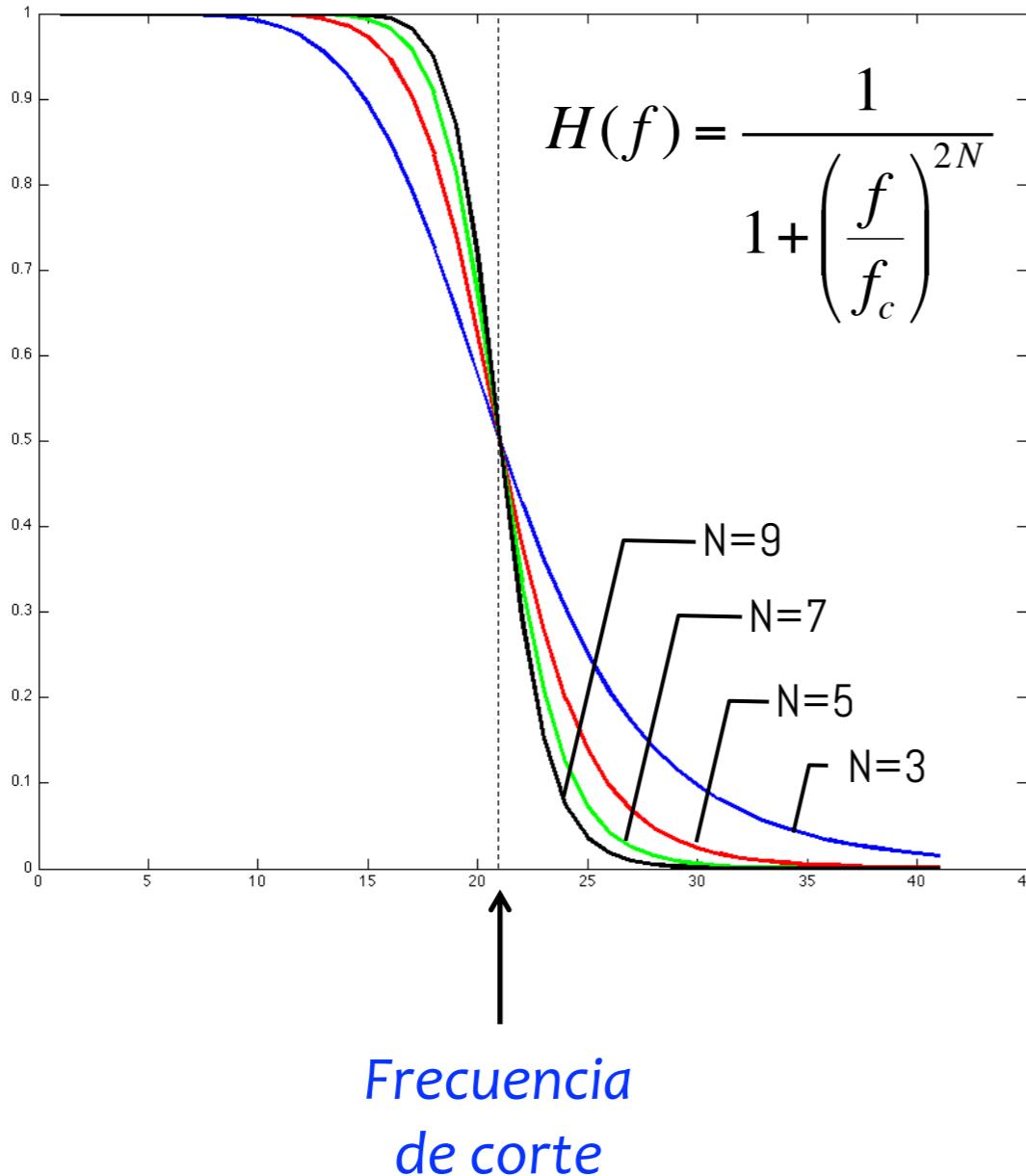
S = np.fft.ifft2(np.fft.fftshift(BF))
out = cv2.normalize(abs(S), None, 0.0, 1.0,
                    cv2.NORM_MINMAX)

cv2.imshow('Salida', out)
cv2.waitKey(0)
```

- Mejoramiento en la frecuencia
  - Convolución y Filtrado
  - Filtro Ideal
  - Filtro Gaussiano
  - Filtro Butterworth
  - Filtro Homomórfico



- ▶ Filtro pasabajos Butterworth (1 dimensión)



```
import numpy as np
import matplotlib.pyplot as plt
fc = 20 # Frecuencia de corte
N = 9 # orden del filtro

m = np.linspace(0, 2*fc, 2*fc)
H = 1/ (1+(np.power(m/fc,2*N)))

plt.plot(H)
plt.show()
```

A medida que el orden del filtro aumenta (N) la caída del filtro se hace mayor formando una recta entre el valor alto y el valor mínimo.

Esto genera la aparición de los efectos indeseados del filtro ideal. Por ello debemos emplear valores menores.

- ▶ Filtro **pasabajos** Butterworth (2 dimensiones)

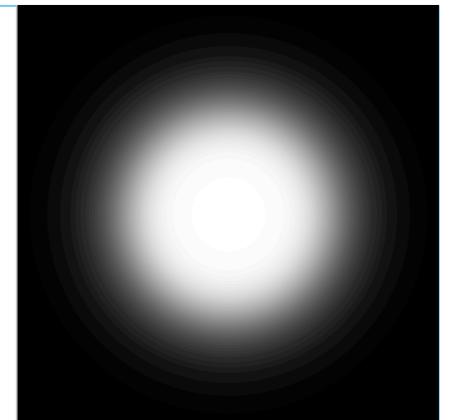
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

fc = 20 # Frecuencia de corte
N = 9 # orden del filtro

x = np.linspace(-2*fc, 2*fc, 4*fc+1)
y = np.linspace(-2*fc, 2*fc, 4*fc+1)
X,Y = np.meshgrid(x,y)

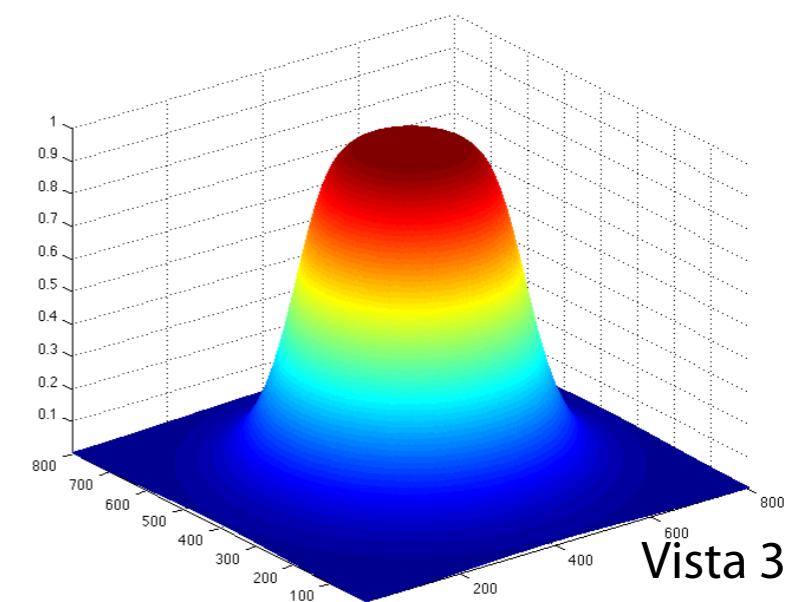
Z = 1/ (1+(np.power(np.sqrt( X**2+Y**2 )/fc,2*N) ))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap=cm.Spectral)
plt.show()
```



Vista 2D del filtro

$$H(x,y) = \frac{1}{1 + \left( \frac{\sqrt{x^2 + y^2}}{f_c} \right)^{2N}}$$

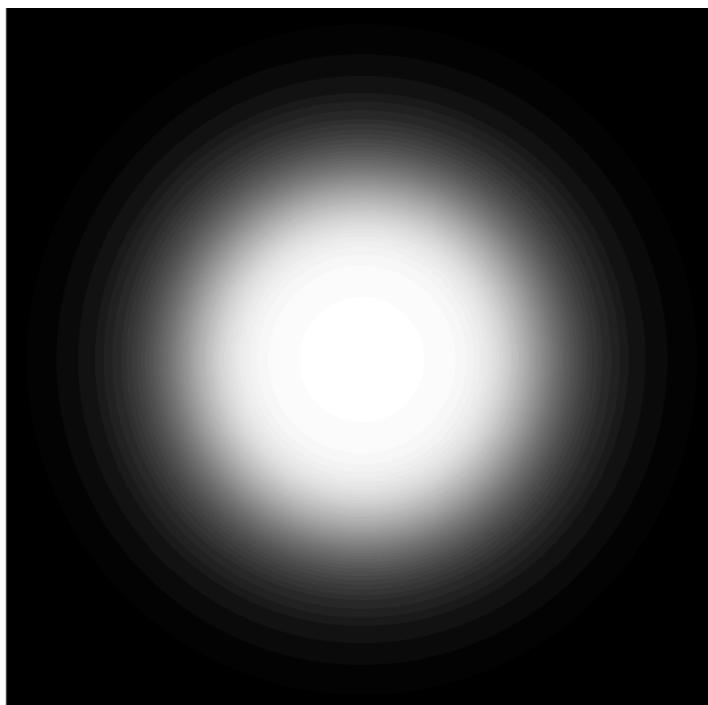


Vista 3D del filtro

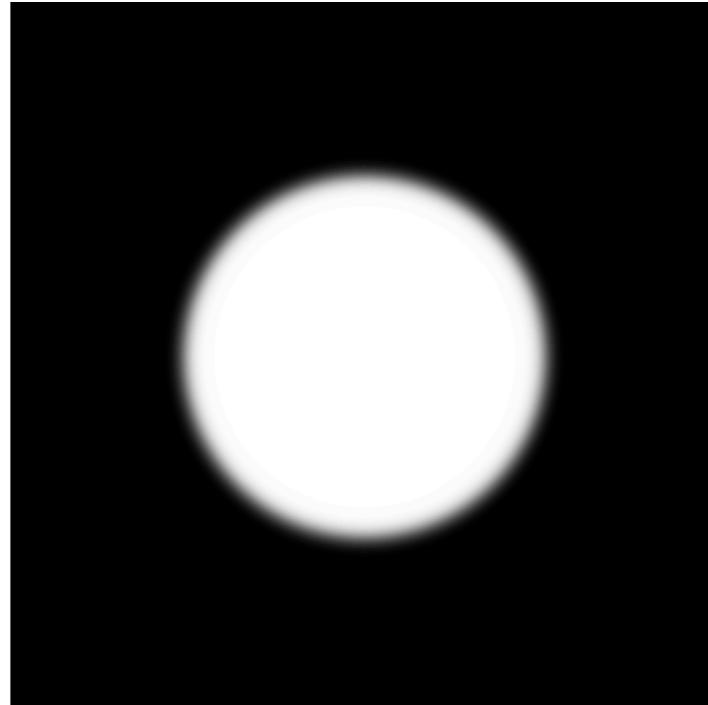


- ▶ Filtro **pasabajos** Variaciones del valor N

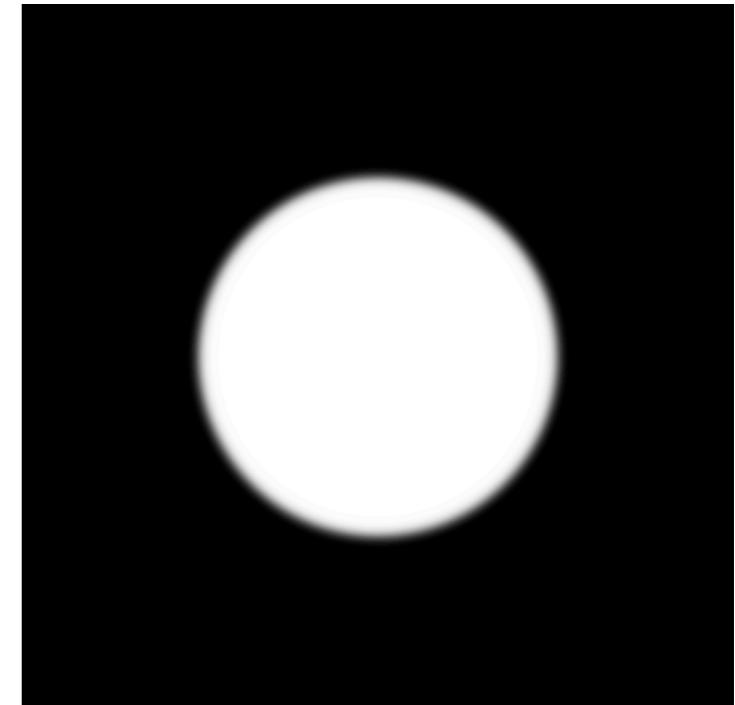
N=3



N=18



N=28

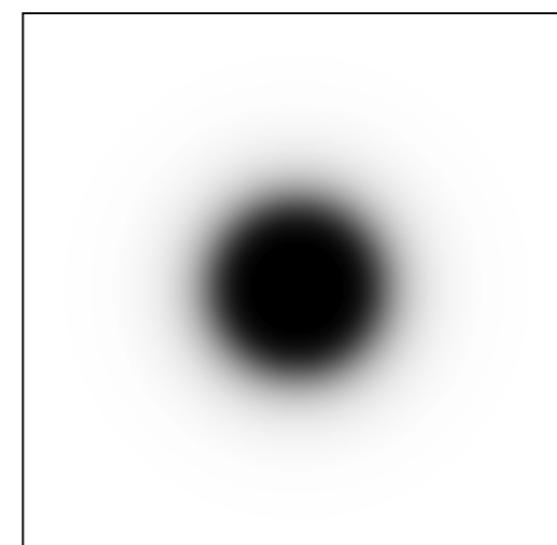
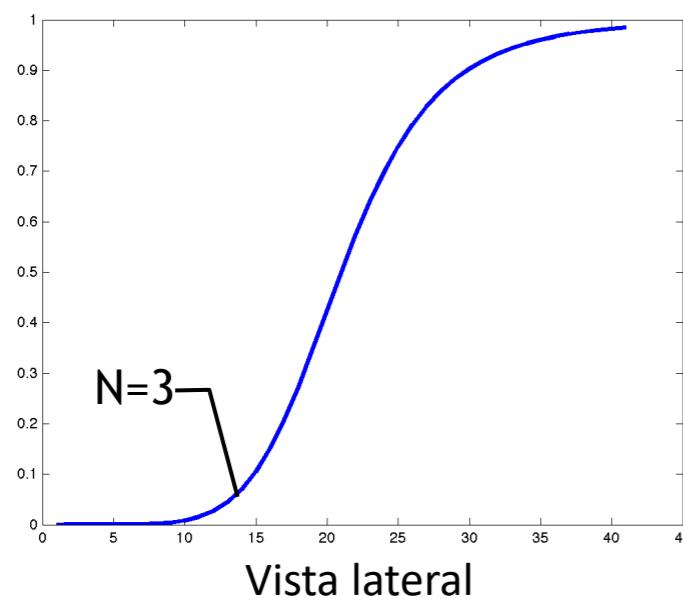


- ▶ Filtro pasaaltos Butterworth (2 dimensiones)

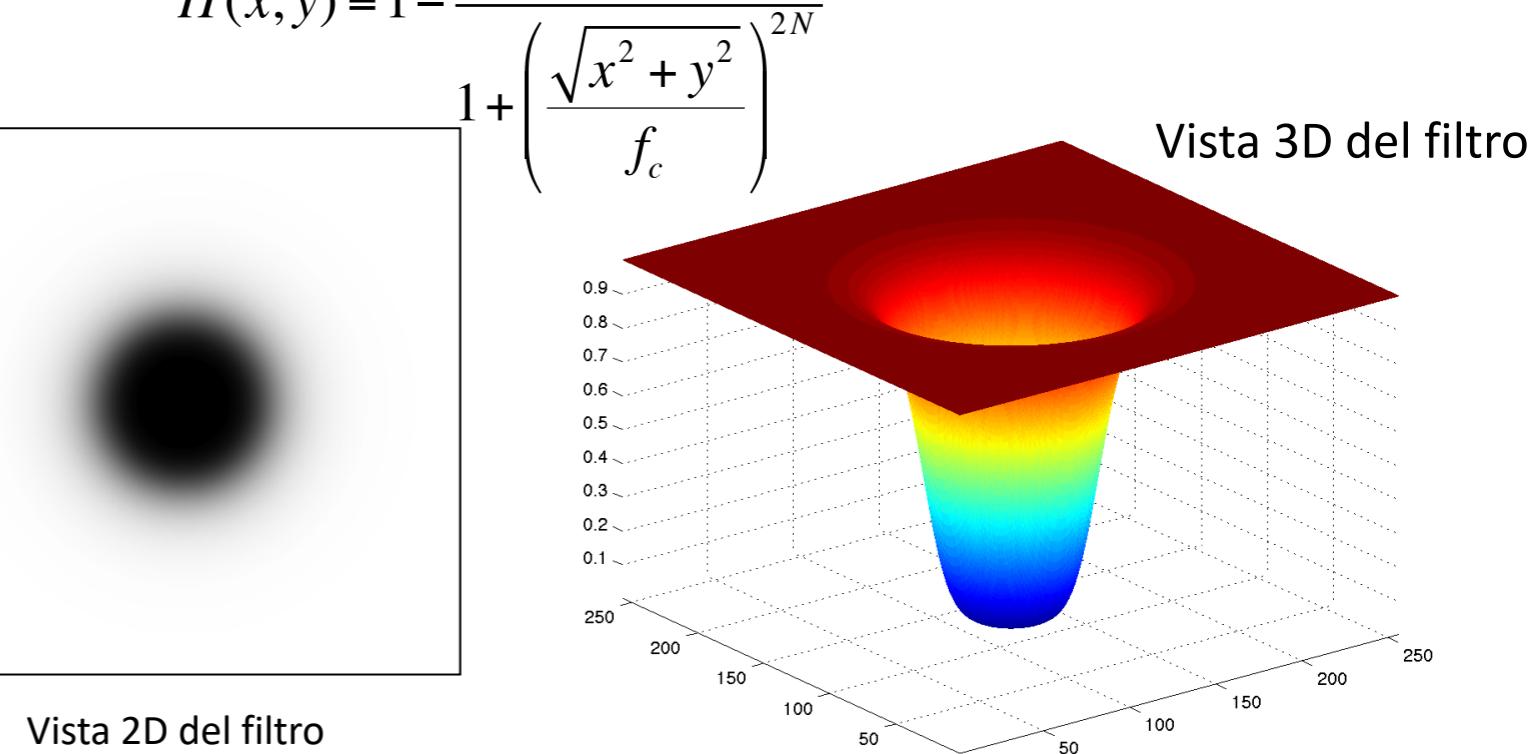
```
fc = 28 # Frecuencia de corte
N = 3 # orden del filtro
```

```
x = np.linspace(-2*fc, 2*fc, 4*fc+1)
y = np.linspace(-2*fc, 2*fc, 4*fc+1)
X,Y = np.meshgrid(x,y)
```

```
Z = 1 - 1/ (1+(np.power(np.sqrt( X**2+Y**2 )/fc,2*N)))
```

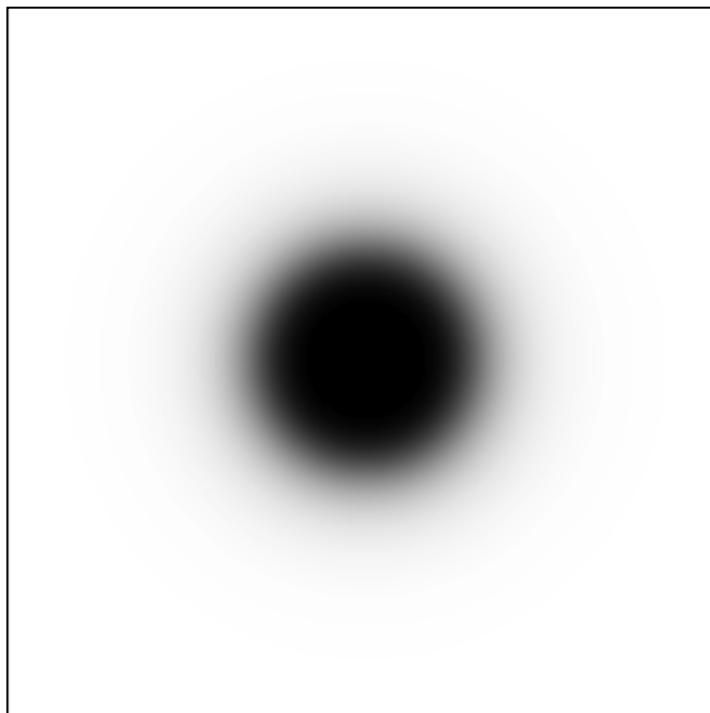


$$H(x,y) = \frac{1}{1 + \left( \frac{\sqrt{x^2 + y^2}}{f_c} \right)^{2N}}$$

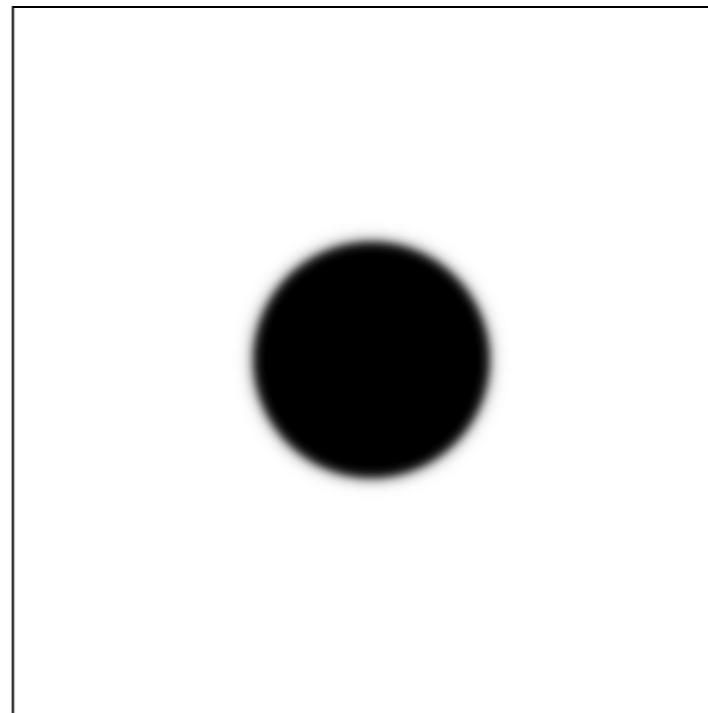


- **Filtro pasaalto Variaciones del valor N**

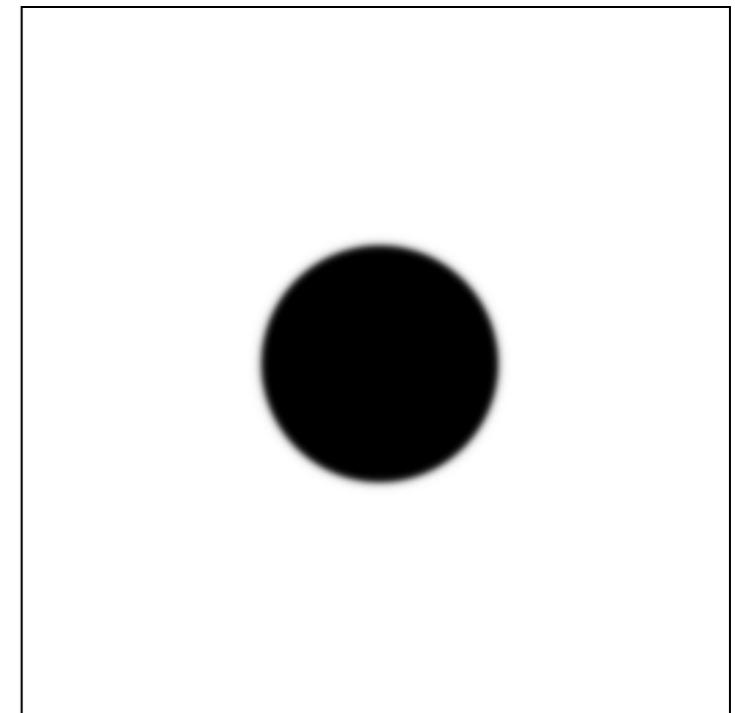
N=3



N=18



N=28



- ▶ Filtro **pasabanda** Butterworth (2 dimensiones)

```

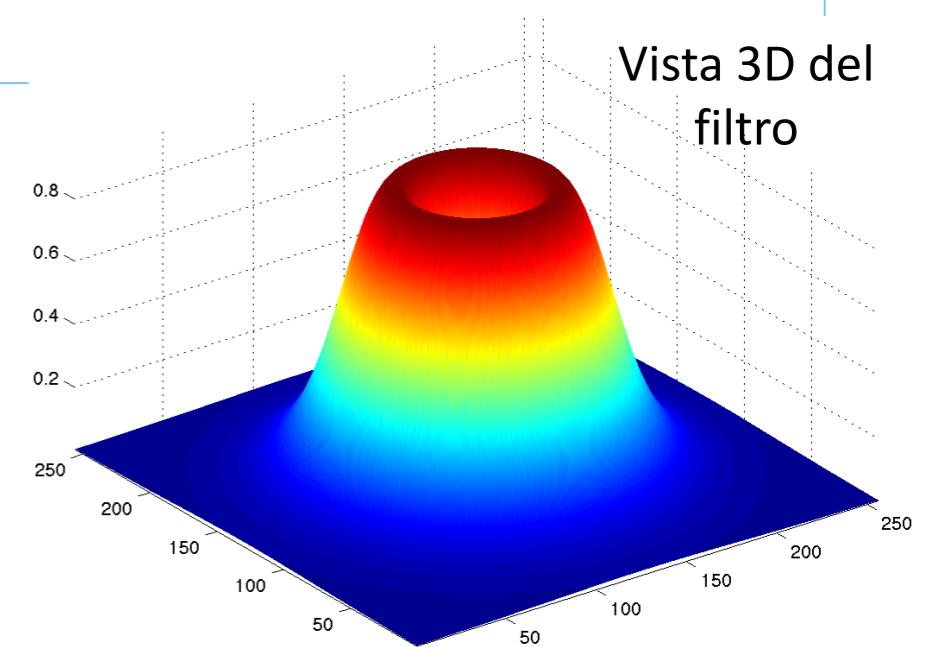
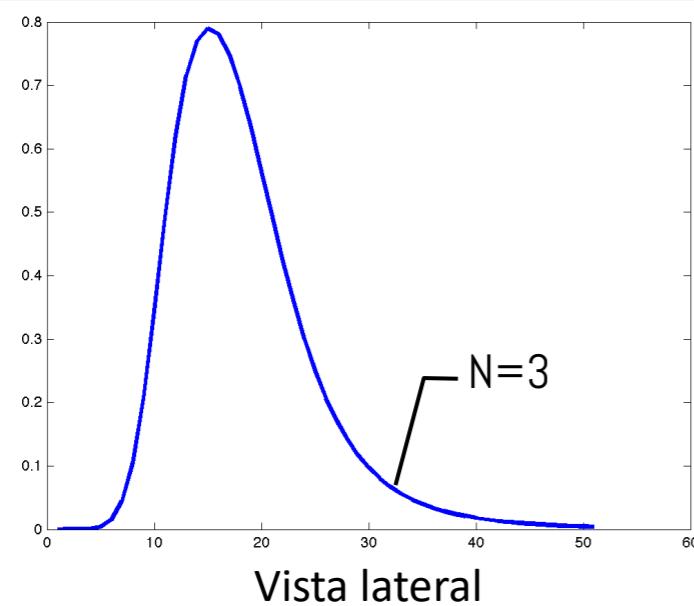
fa = 10 # Frecuencia de corte
fc = 28 # Frecuencia de corte
fb = 30 # Frecuencia de corte
N = 3 # orden del filtro

x = np.linspace(-2*fc, 2*fc, 256)
y = np.linspace(-2*fc, 2*fc, 256)
X,Y = np.meshgrid(x,y)

H1 = 1 / (1+(np.power(np.sqrt( X**2+Y**2 )/fa,2*N)))
H2 = 1- 1 / (1+(np.power(np.sqrt( X**2+Y**2)/fb,2*N)))

Z = H1 * H2

```



## ▶ Filtro pasabanda Butterworth

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

img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

F = np.fft.fft2(gray)
FS = np.fft.fftshift(F)

fa=10 # Frecuencia de corte
fc=28 # Frecuencia de corte
fb=30 # Frecuencia de corte

N= 3 # orden del filtro

x= np.linspace(-2*fc, 2*fc, 256)
y= np.linspace(-2*fc, 2*fc, 256)
X, Y = np.meshgrid(x,y)

H1 = 1/ (1+(np.sqrt(np.power(X,2)+np.power(Y,2))/fa,2*N)))
H2 = 1- 1/ (1+(np.sqrt(np.power(X,2)+np.power(Y,2))/fb,2*N)))

Z = H1 * H2
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap=cm.Spectral)
plt.show()

BF = Z * FS

S= np.fft.ifft2(np.fft.fftshift(BF))
out = cv2.normalize(abs(S), None, 0.0, 1.0, cv2.NORM_MINMAX)

cv2.imshow('Salida', out)
cv2.waitKey(0)
```



- Mejoramiento en la frecuencia
  - Convolución y Filtrado
  - Filtro Ideal
  - Filtro Gaussiano
  - Filtro Butterworth
  - Filtro Homomórfico



- ▶ Idea general:
  - La imagen es el producto de la Iluminación con Reflectancia

$$f(x, y) = i(x, y) \cdot r(x, y)$$

- El Filtro homomórfico opera sobre cada una de las componentes, por ello empleamos el logaritmo (para separar ambas componentes)

$$\ln(f(x, y)) = \ln(i(x, y)) + \ln(r(x, y))$$

- Al emplear la transformada de Fourier sobre la imagen anterior obtenemos

$$F(\ln(f(x, y))) = F(\ln(i(x, y))) + F(\ln(r(x, y)))$$

► Idea general:

- En otras palabras, la transformada de Fourier del logaritmo de la imagen corresponde a la suma de la iluminancia y reflectancia en la frecuencia.

$$F(\ln(f(x,y))) = F(\ln(i(x,y))) + F(\ln(r(x,y)))$$

$$Z(u,v) = F_i(u,v) + F_r(u,v)$$

- Lo anterior significa que si aplicamos un filtro para la imagen podemos aplicarlo a ambos canales.

$$Z(u,v) \cdot H(u,v) = F_i(x,y) \cdot H(u,v) + F_r(x,y) \cdot H(u,v)$$

↑                      ↑                      ↑  
Filtro                  Filtro                  Filtro

► Idea general:

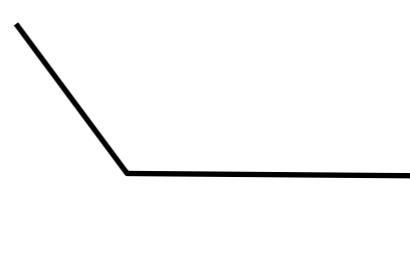
- En el siguiente paso, al aplicar la transformada inversa de fourier

$$\begin{aligned}s(x, y) &= F^{-1}(Z(u, v)H(u, v)) \\&= F^{-1}\{F_i(x, y)H(u, v)\} + F^{-1}\{F_r(x, y)H(u, v)\} \\&= i'(x, y) + r'(x, y)\end{aligned}$$

- Donde  $i$ , y  $r$  corresponden a las componentes de iluminancia y reflectancia. Dado que en el primer paso aplicamos el logaritmo, el paso final consiste en aplicar la función exponencial.

$$g(x, y) = e^{s(x, y)} = e^{i'(x, y) + r'(x, y)}$$

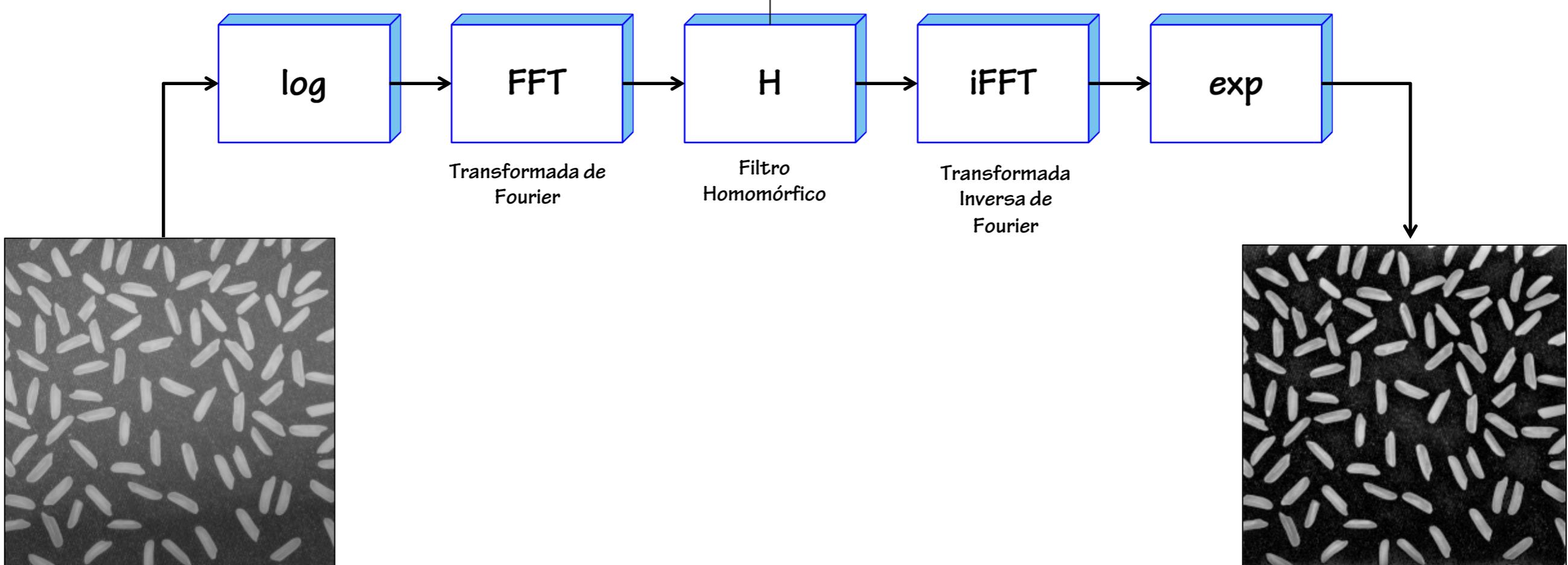
$$g(x, y) = i_0(x, y)r_0(x, y)$$



Finalmente obtenemos las componentes de iluminancia y reflectancia modificadas

## ▶ Proceso general:

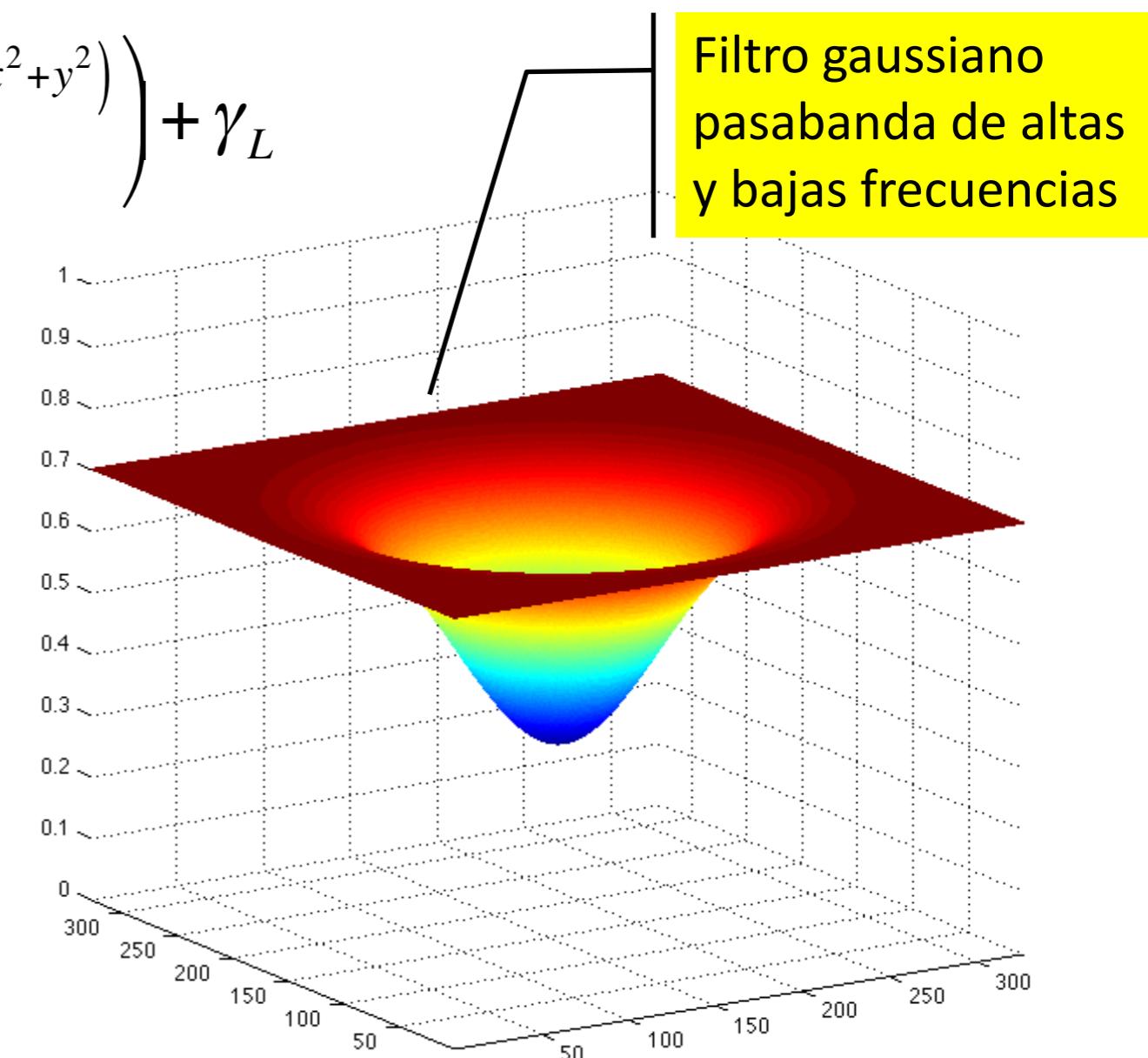
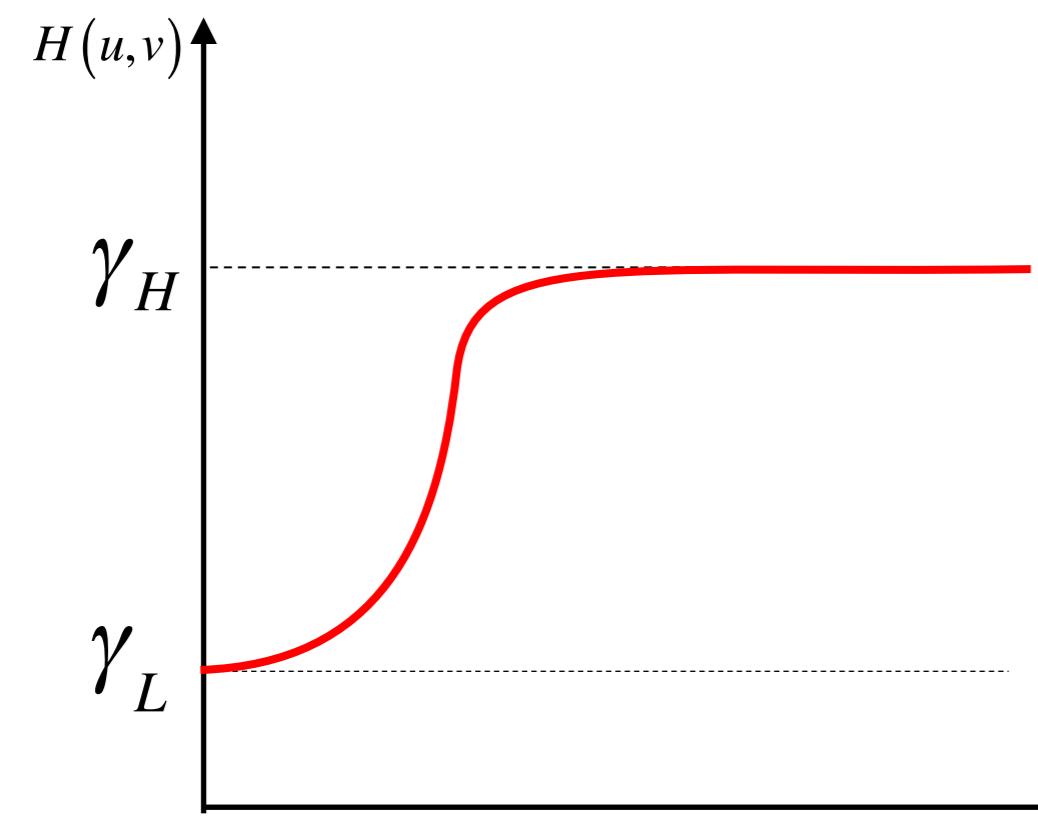
- ✓ Gracias al filtro homomórfico, tenemos más control sobre las componente de alta y baja frecuencias.
- ✓ Las componentes de iluminancia representan las componentes de baja frecuencia y la reflectancia las de alta frecuencia.



## ▶ Creando el filtro

- El objetivo es diseñar un filtro  $H$  que modifique las niveles de Iluminancia y de reflectancia al mismo tiempo. Un filtro que podemos emplear puede ser.

$$H(u, v) = (\gamma_H - \gamma_L) \cdot \left(1 - e^{-c \cdot (x^2 + y^2)}\right) + \gamma_L$$



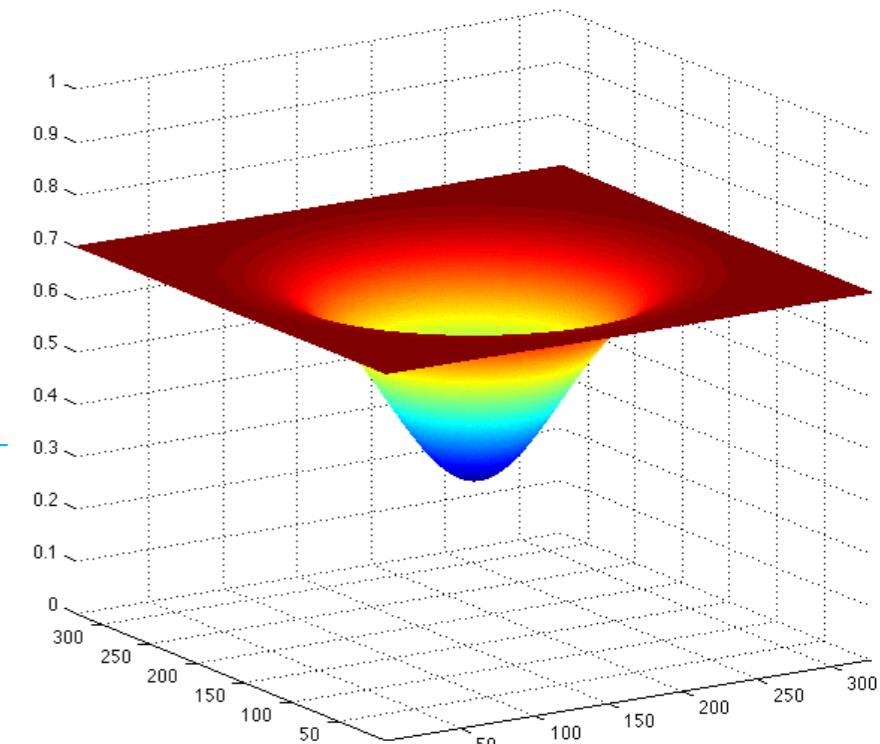
## ▶ Creando el filtro

- El objetivo es diseñar un filtro  $H$  que modifique las niveles de Iluminancia y de reflectancia al mismo tiempo. Un filtro que podemos emplear puede ser.

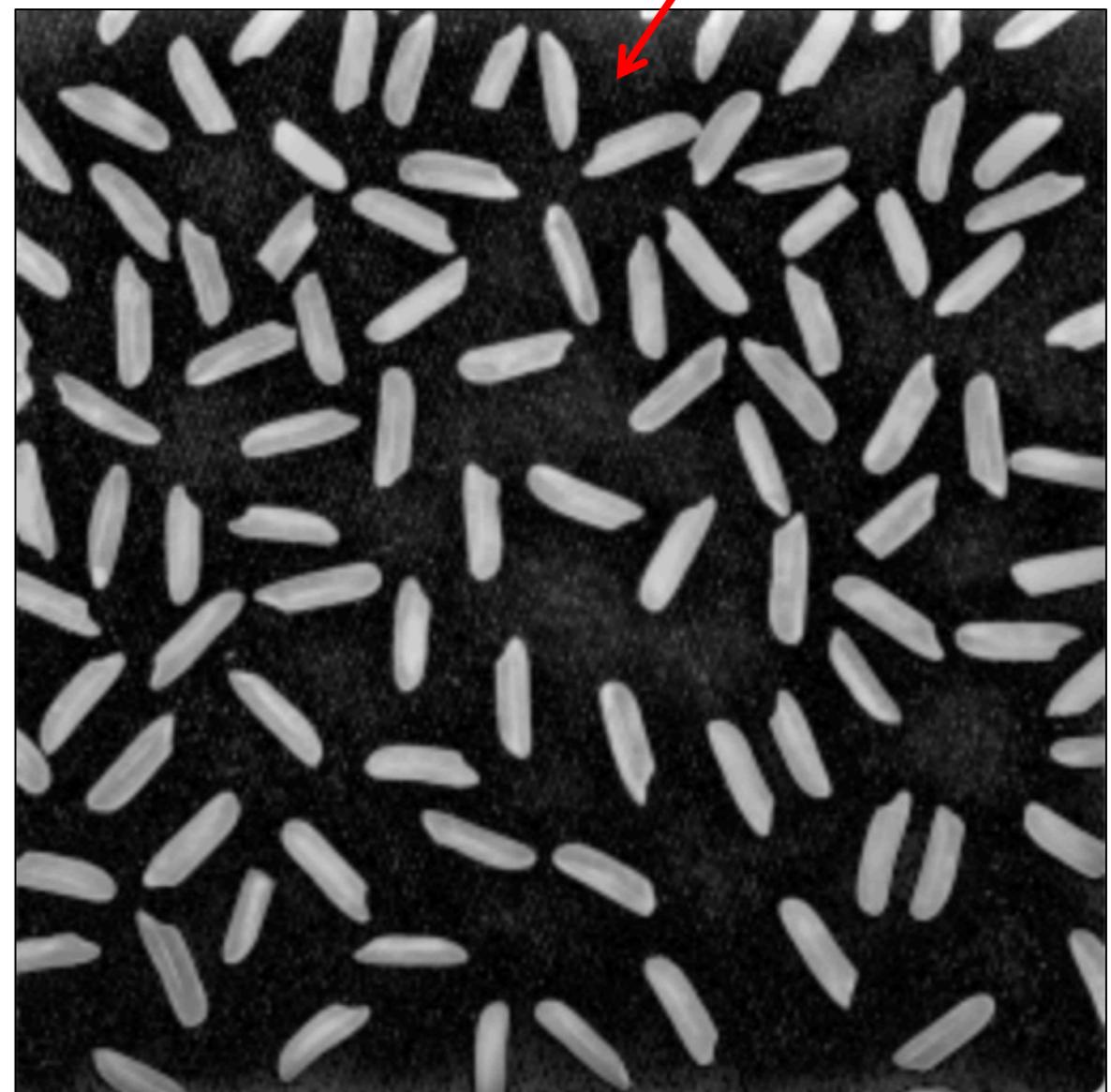
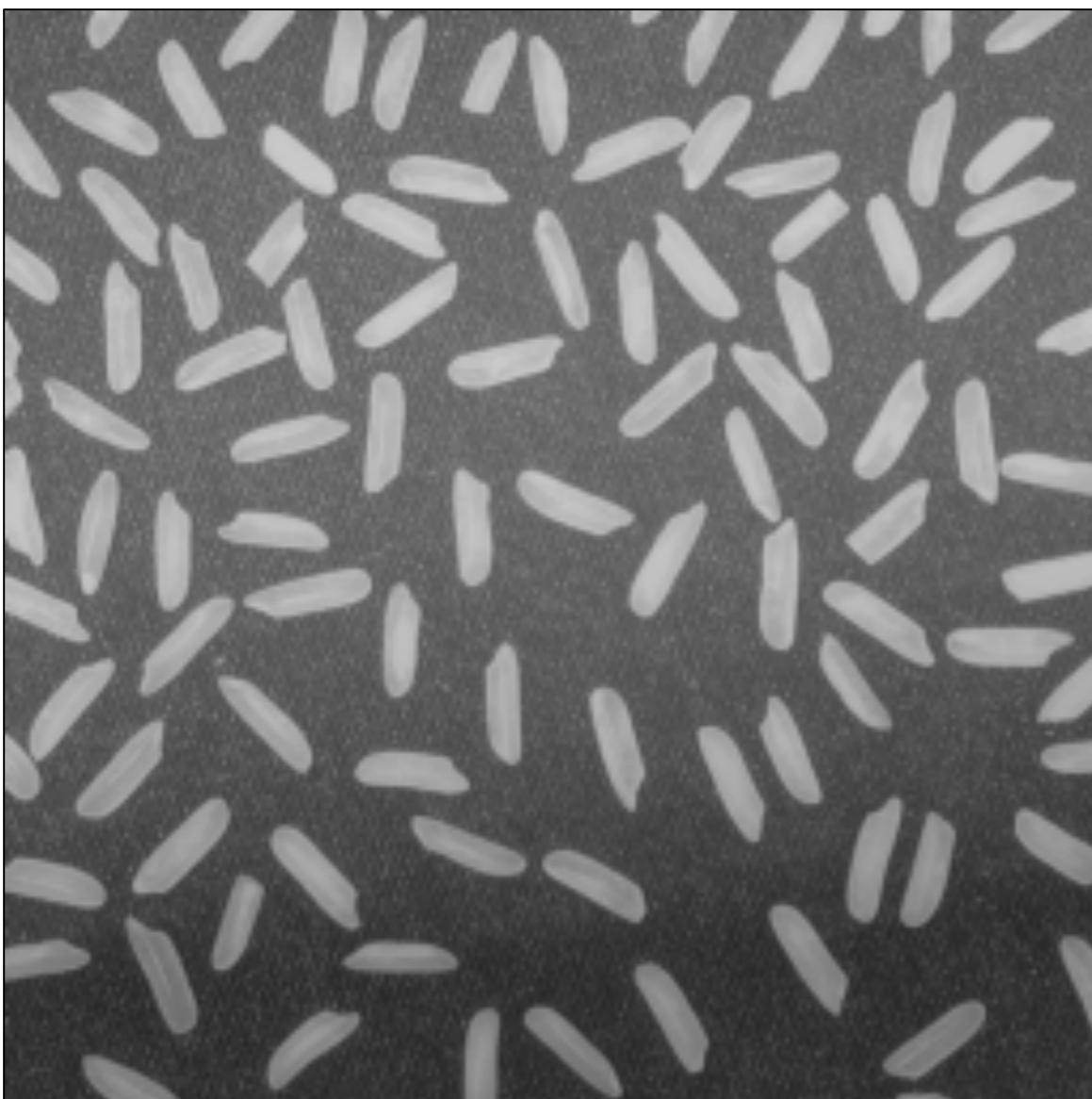
$$H(u, v) = (\gamma_H - \gamma_L) \cdot \left( 1 - e^{-c \cdot (x^2 + y^2)} \right) + \gamma_L$$

```
m = 256
YH = 0.9
YL = 0.2
c = 0.005
vector = np.linspace(-m,m, m)
X, Y = np.meshgrid(vector,vector)

Z = (YH-YL) * (1-np.exp(-c* (X**2+Y**2) ))+YL
```



- ▶ Filtro Homomórfico en acción



Notar el fondo de la imagen

## ▶ Filtro Homomórfico en acción

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

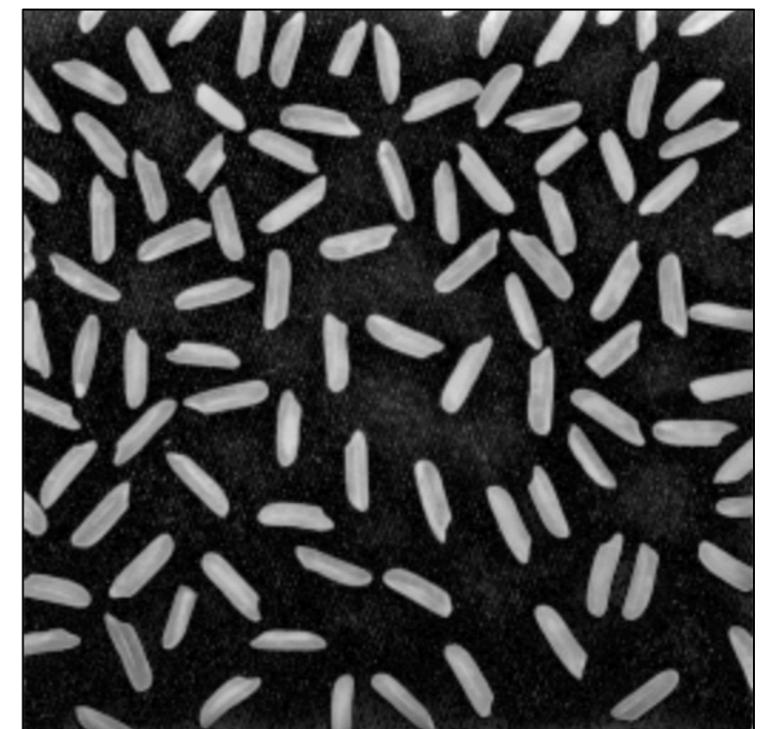
img = cv2.imread('rice.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
F = np.fft.fft2(gray)
FS = np.fft.fftshift(F)

m = 256
YH = 0.9
YL = 0.2
c = 0.005
vector = np.linspace(-m,m, m)
X, Y = np.meshgrid(vector, vector)
Z = (YH-YL) * (1-np.exp(-c* (np.power(X,2)+np.power(Y,2))))+YL

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap=cm.Spectral)
plt.show()

BF = Z * FS
S= np.fft.ifft2(np.fft.fftshift(BF))
out = cv2.normalize(abs(S), None, 0.0, 1.0, cv2.NORM_MINMAX)

cv2.imshow('Salida', out)
cv2.waitKey(0)
```





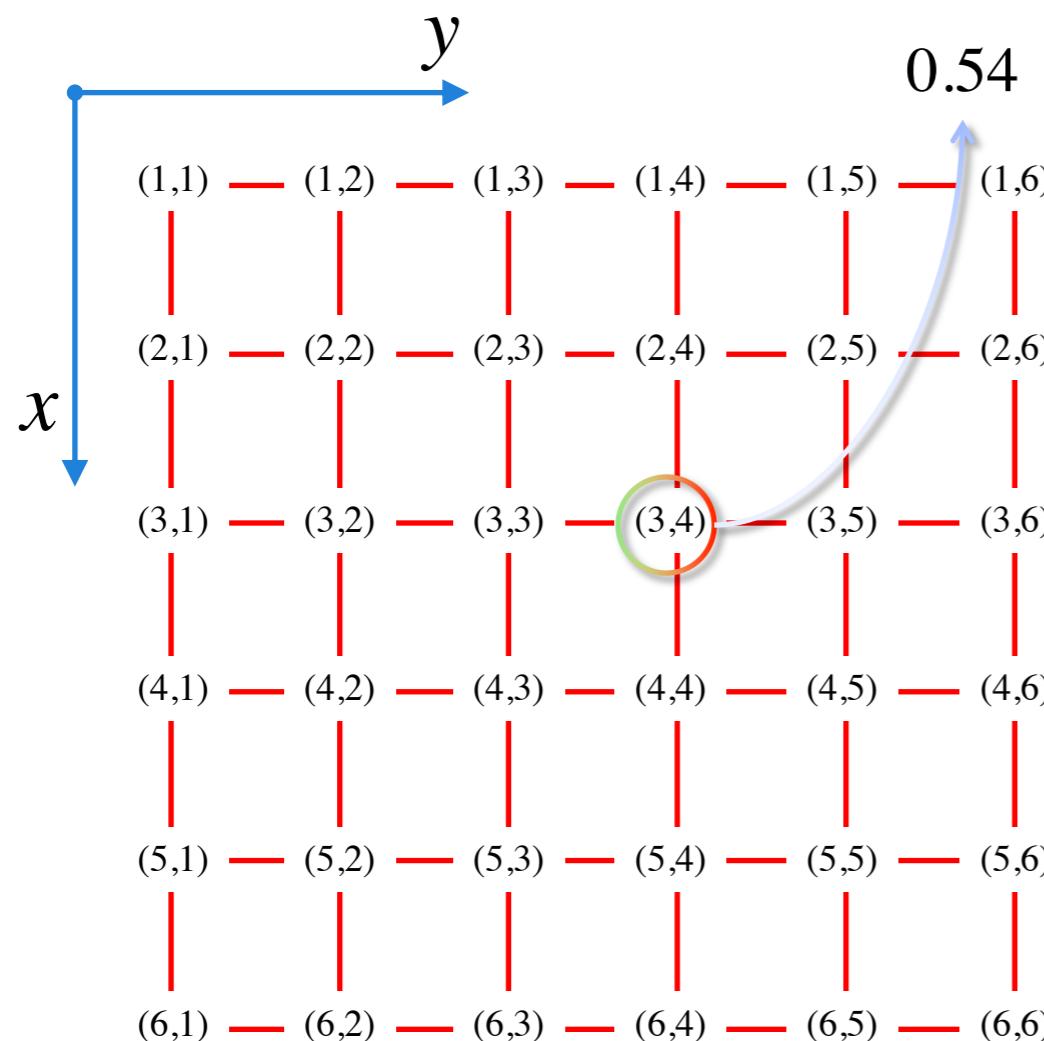
# Anexo

Comando Meshgrid



▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.



- Suponga que usted desea evaluar la siguiente función  $f(x,y)$  entre 1 y 6 para  $x$  e  $y$ :

$$f(x, y) = \cos(x - y) \quad \forall x = [1, \dots, 6]$$

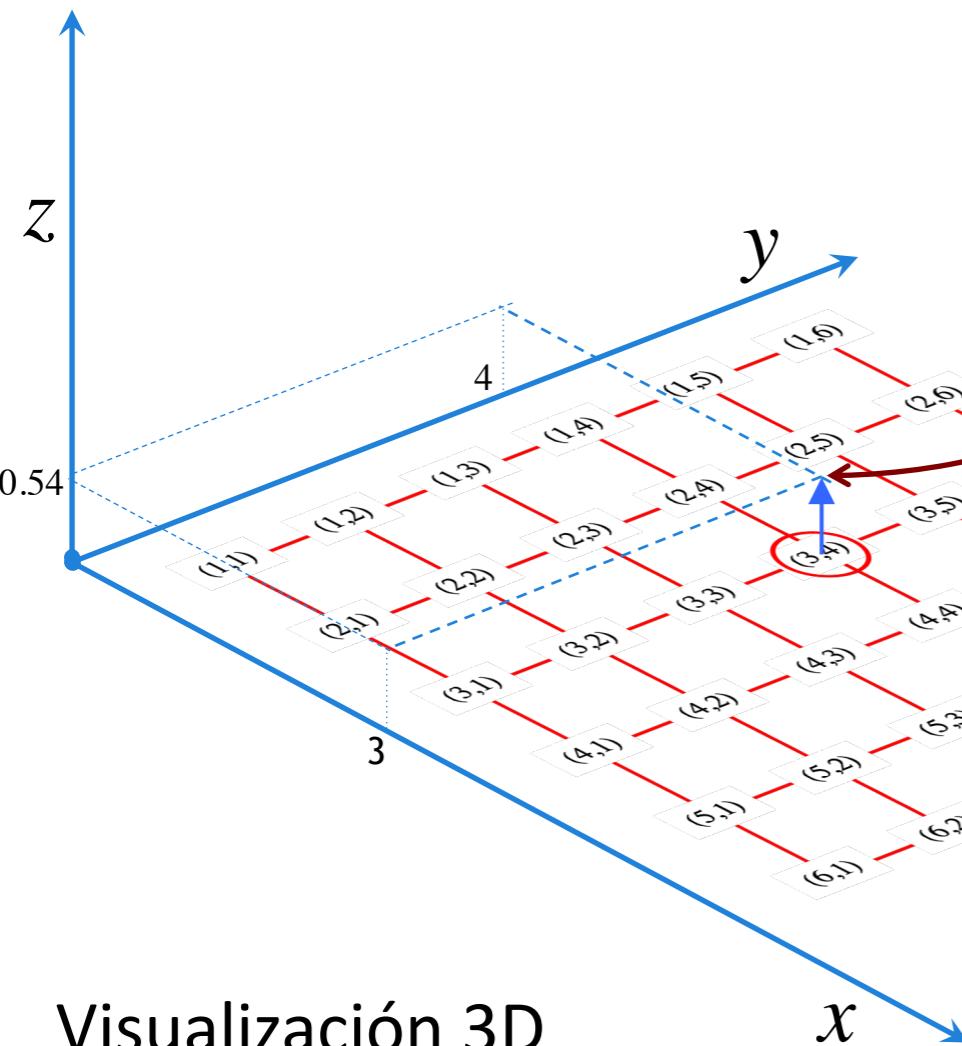
$$\forall y = [1, \dots, 6]$$

- Para cada punto de la matriz debe evaluar la ecuación, por ejemplo:

$$f(3, 4) = \cos(3 - 4) = 0.54$$

- ▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.



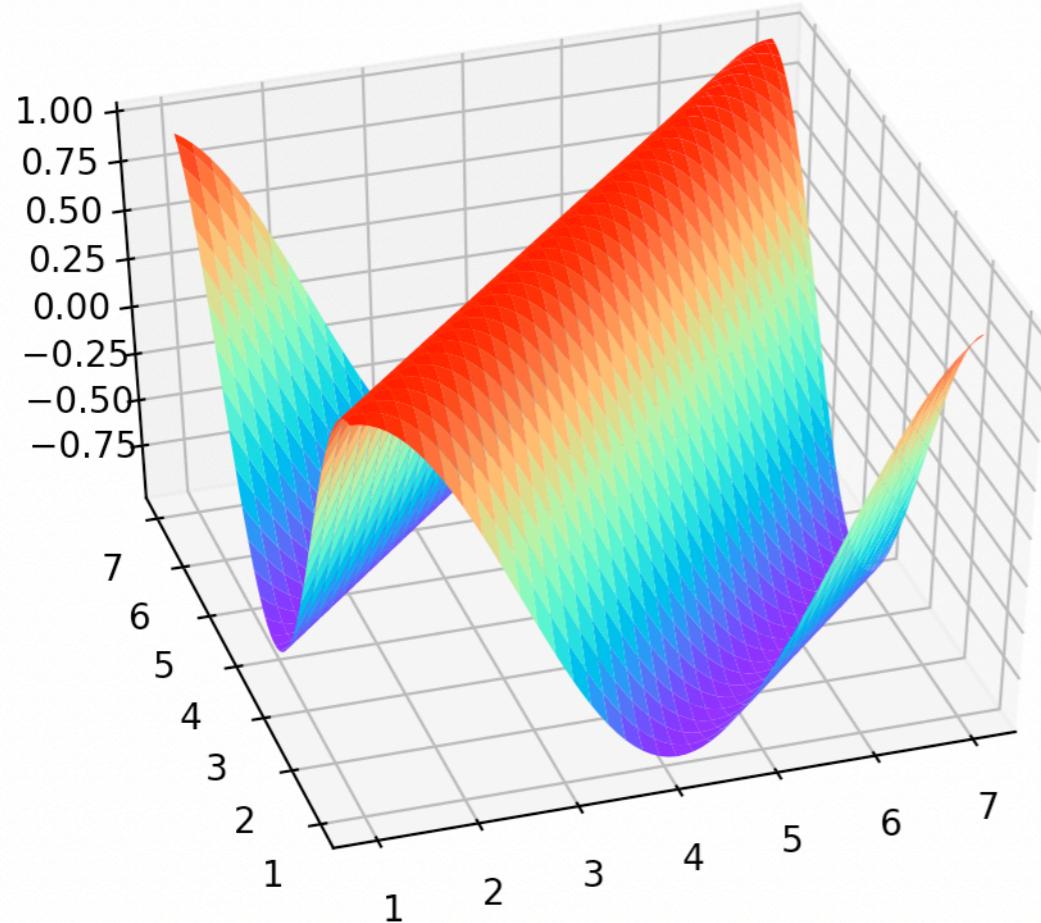
- Si visualizamos en el eje Z el resultado, este corresponde a 0.54

$$f(3,4) = \cos(3 - 4) = 0.54$$

- Como observamos, cada par ordenado  $(x, y)$  genera un resultado. Por lo tanto debemos calcular para cada uno de los puntos de la matriz un valor.

▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.



- Lo anterior puede ser calculado con doble ciclo `for`. De esta forma en el ciclo interno vamos generando cada coordenada que será evaluada en nuestra función

$$f(x, y) = \cos(x - y) \quad \forall x = [1, \dots, 6] \\ \forall y = [1, \dots, 6]$$

En Python (versión simple y lenta)

```
for x in range(1,7):
    for y in range(1,7):
        f[x,y]=np.cos(x-y)
    end
end
```

▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.

En Python (con meshgrid)

```
import numpy as np
x = np.linspace(1,7,7)
y = np.linspace(1,7,7)
X,Y = np.meshgrid(x,y)
```

- Python provee una función que determina todos los puntos  $x, y$  que requerimos para evaluar la función.

`X =`

1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

`Y =`

1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6

Los índices de estas matrices serán ocupados en nuestra función. Si observan cada punto corresponde a un par ordenado

▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.

### En Python (con meshgrid)

```
import numpy as np
x = np.linspace(1,7,7)
y = np.linspace(1,7,7)
X,Y = np.meshgrid(x,y)
P = X-Y
```

- Dado que X e Y son matrices, podemos operar con ellas (sumar, restar multiplicar, dividir, etc.)

`X =`

1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

`Y =`

1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6

`P=`

[[ 0.	1.	2.	3.	4.	5.	6.]
[-1.	0.	1.	2.	3.	4.	5.]
[-2.	-1.	0.	1.	2.	3.	4.]
[-3.	-2.	-1.	0.	1.	2.	3.]
[-4.	-3.	-2.	-1.	0.	1.	2.]
[-5.	-4.	-3.	-2.	-1.	0.	1.]
[-6.	-5.	-4.	-3.	-2.	-1.	0.]]

▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.

## En Python (con meshgrid)

```
import numpy as np
x = np.linspace(1, 7, 7)
y = np.linspace(1, 7, 7)
X, Y = np.meshgrid(x, y)
P = X - Y
f = np.cos(P)
```

- También podemos aplicar una función sobre una matriz, por ejemplo, coseno, seno, exponencial, tangente, etc. Esto se aplica a cada valor de la matriz

P =

```
[[ 0.  1.  2.  3.  4.  5.  6.]
 [-1.  0.  1.  2.  3.  4.  5.]
 [-2. -1.  0.  1.  2.  3.  4.]
 [-3. -2. -1.  0.  1.  2.  3.]
 [-4. -3. -2. -1.  0.  1.  2.]
 [-5. -4. -3. -2. -1.  0.  1.]
 [-6. -5. -4. -3. -2. -1.  0.]]
```

*coseno* →

f =

```
[[ 1.      0.54   -0.42  -0.99  -0.65   0.28   0.96 ]
 [ 0.54    1.      0.54   -0.42  -0.99  -0.65   0.28 ]
 [-0.42   0.54    1.      0.540  -0.42  -0.99  -0.65 ]
 [-0.99  -0.42    0.54    1.      0.54   -0.42  -0.99 ]
 [-0.65  -0.99   -0.42    0.54    1.      0.54   -0.42 ]
 [ 0.28  -0.65   -0.99   -0.42    0.54    1.      0.54 ]
 [ 0.96   0.28   -0.65  -0.99  -0.42    0.54    1.     ]]
```

- ▶ El comando `surf` permite plotear tridimensionalmente cada valor de la matriz. En el eje Z queda se muestra el valor de cada punto de la matriz

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
```

*evaluación de la función*

```
x = np.linspace(1,7,70)
y = np.linspace(1,7,70)

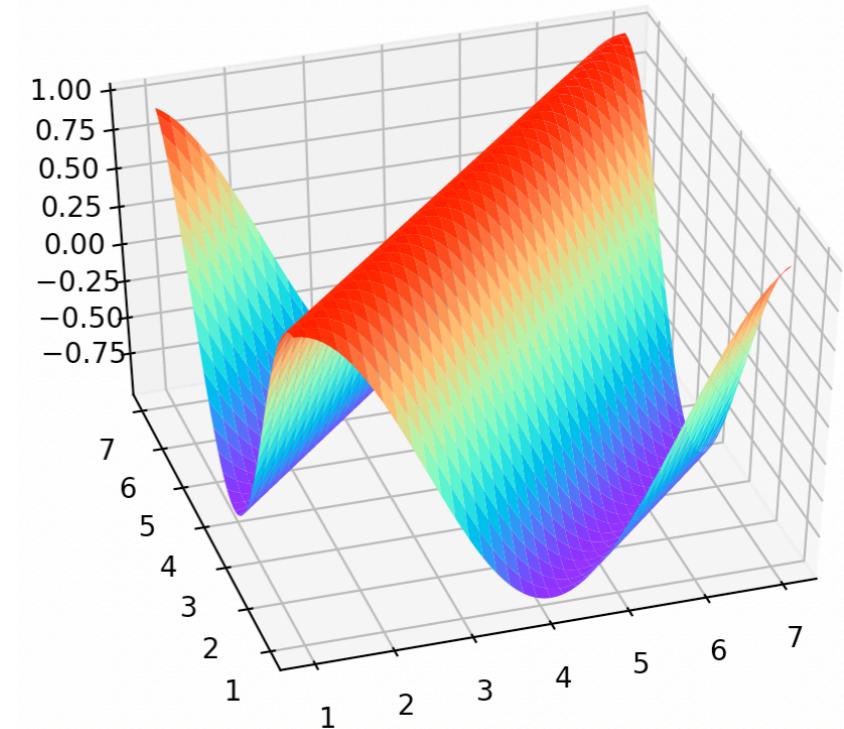
X,Y = np.meshgrid(x,y)
P = X-Y
f = np.cos(P)
```

*gráfica de la matriz f*

```
fig = plt.figure()
ax = fig.gca(projection='3d')

surf = ax.plot_surface(X, Y, f,
                      cmap=cm.rainbow,
                      linewidth=0, antialiased=True)

plt.show()
```



▶ `X, Y = np.meshgrid(x, y)`

Es un comando de numpy que permite crear la malla completa de puntos  $x, y$  que pueden ser evaluados en otra función.

### En Python (MUY LENTO)

```
for x in range(1,7):
    for y in range(1,7):
        f[x,y]=np.cos(x-y)
    end
end
```



Obtenemos el mismo resultado, pero la versión con `for` es lenta

### En Python (EFICIENTE)

```
import numpy as np
x = np.linspace(1,7,7)
y = np.linspace(1,7,7)
X,Y = np.meshgrid(x,y)
P = X-Y
f = np.cos(P)
```

Evite ocupar ciclos `for` en Python ya que son muy lentos. Ocupe operaciones matriciales que son eficientes y rápidas

