



UAI

UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS



iUAI
UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERIA Y CIENCIAS

MDS²⁰¹⁹ PROCESAMIENTO
DE IMÁGENES

MEJORAMIENTO EN LA FRECUENCIA

Miguel Carrasco
miguel.carrasco@uai.cl
1er Semestre 2020

- Mejoramiento en la frecuencia
 - Preliminar
 - Señales 1D
 - Señales 2D (imágenes)
 - Aplicaciones

▶ Jean-Baptiste Joseph Fourier (1768-1830)

- Fue un matemático y Físico Francés que desarrolló un método para decomponer funciones periódicas en funciones trigonométricas convergentes (llamadas series de Fourier)

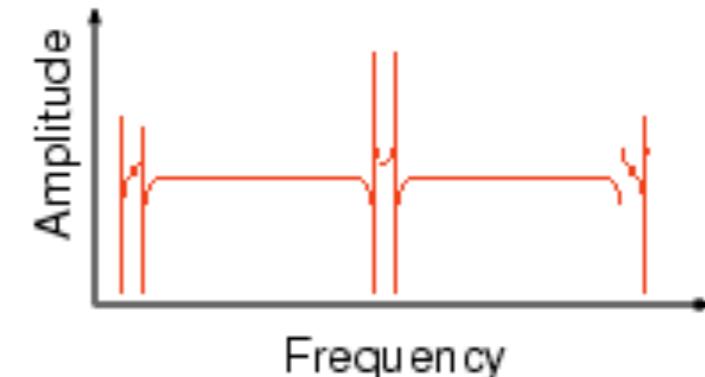
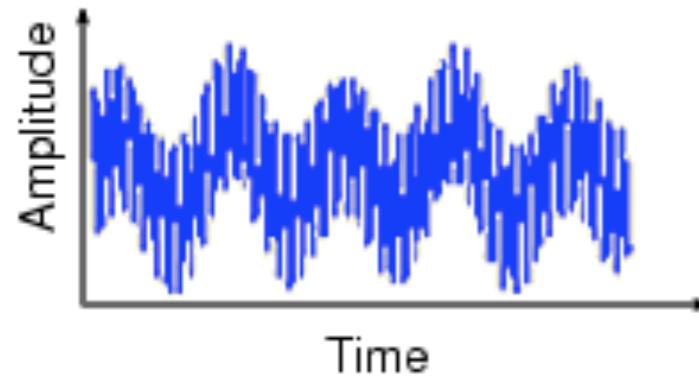
$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos \frac{2n\pi}{T} t + b_n \sin \frac{2n\pi}{T} t \right]$$

- Gracias al método desarrollado fue posible modelar las ecuaciones del calor, muy empleadas para describir otros fenómenos en matemáticas.
- Además fue el primero en descubrir el fenómeno del efecto invernadero (a través de ecuaciones diferenciales)



Jean-Baptiste Joseph Fourier (1768-1830)

► Idea general



- En el análisis de señales, la transformada de Fourier es una de las herramientas más empleadas en matemáticas ya que permite transformar una señal

basada-en-el-tiempo a una señal **basada-en-la-frecuencia**

El mayor problema es que una vez en la frecuencia, la información del tiempo se pierde. Pero eso no es problema en el procesamiento de imágenes, ya que no es relevante el tiempo. La **señal es el color en la imagen**, y **el tiempo es su posición en el espacio**.

- ▶ Transformada Discreta de Fourier (DFT)

$$X_r = \sum_{n=0}^{N-1} x_n \cdot e^{-i \frac{2\pi}{N} \cdot r \cdot n}$$

Para $n = 0 \dots N - 1$

The diagram shows the DFT formula with several annotations:

- A blue box labeled "número imaginario" has a line pointing to the term i in the exponent.
- A blue box labeled "Vector de largo N" has a line pointing to the summation index n .
- A blue box labeled "Transformada de Fourier" has a line pointing to the entire formula.

A blue callout box contains the text: "La transformada de Fourier transforma una señal en el tiempo (o espacio) a una en la frecuencia."

- Veamos un ejemplo

$x =$	5	4	2	1	7	9
	0	1	2	3	4	5

Supongamos una señal x con 6 valores

Calculemos la primera componente, es decir con $r=0$

$$X_0 = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} \cdot 0 \cdot n}$$

Primera componente

$$X_0 = 5 \cdot e^{-\frac{2\pi i}{6} \cdot 0 \cdot 0} + 4 \cdot e^{-\frac{2\pi i}{6} \cdot 0 \cdot 1} + 2 \cdot e^{-\frac{2\pi i}{6} \cdot 0 \cdot 2} + 1 \cdot e^{-\frac{2\pi i}{6} \cdot 0 \cdot 3} + 7 \cdot e^{-\frac{2\pi i}{6} \cdot 0 \cdot 4} + 9 \cdot e^{-\frac{2\pi i}{6} \cdot 0 \cdot 5}$$

$$X_0 = 5 + 4 + 2 + 1 + 7 + 9 = 28$$

- Veamos un ejemplo

$x =$	5	4	2	1	7	9
	0	1	2	3	4	5

Calculemos la segunda componente, es decir con $r=1$

$$X_1 = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} \cdot 1 \cdot n}$$

Segunda componente

$$X_1 = 5 \cdot e^{-\frac{2\pi i}{6} \cdot 1 \cdot 0} + 4 \cdot e^{-\frac{2\pi i}{6} \cdot 1 \cdot 1} + 2 \cdot e^{-\frac{2\pi i}{6} \cdot 1 \cdot 2} + 1 \cdot e^{-\frac{2\pi i}{6} \cdot 1 \cdot 3} + 7 \cdot e^{-\frac{2\pi i}{6} \cdot 1 \cdot 4} + 9 \cdot e^{-\frac{2\pi i}{6} \cdot 1 \cdot 5}$$

$$X_1 = 6 + 8.66i$$

- Veamos un ejemplo

$x =$	5	4	2	1	7	9
	0	1	2	3	4	5

Calculemos la tercera componente, es decir con $r=2$

$$X_2 = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} \cdot 2 \cdot n}$$

Tercera componente

$$X_2 = 5 \cdot e^{-\frac{2\pi i}{6} \cdot 2 \cdot 0} + 4 \cdot e^{-\frac{2\pi i}{6} \cdot 2 \cdot 1} + 2 \cdot e^{-\frac{2\pi i}{6} \cdot 2 \cdot 2} + 1 \cdot e^{-\frac{2\pi i}{6} \cdot 2 \cdot 3} + 7 \cdot e^{-\frac{2\pi i}{6} \cdot 2 \cdot 4} + 9 \cdot e^{-\frac{2\pi i}{6} \cdot 2 \cdot 5}$$

$$X_2 \approx -5$$

- ▶ Transformada DFT (no óptima)

$$X_r = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi}{N} \cdot r \cdot n}$$



```
def fft(x):
    N = len(x)

    if N <= 1:
        return x
    tmp = np.zeros(N, dtype=np.complex)
    X = np.zeros(N, dtype=np.complex)

    for r in range(0,N):
        for n in range(0,N):
            tmp[n]= x[n]*exp(-2j*pi*r*n/N)
        X[r]= np.sum(tmp)
    return X
```

- ▶ Transformada Inversa Discreta de Fourier (IDFT)

$$x_n = \frac{1}{N} \sum_{r=0}^{N-1} X_r \cdot e^{-j\frac{2\pi}{N} \cdot r \cdot n}$$

Para $n = 0 \dots N - 1$

The diagram illustrates the IDFT formula with callout boxes and arrows:

- A blue box labeled "número imaginario" has an arrow pointing to the term $e^{-j\frac{2\pi}{N} \cdot r \cdot n}$.
- A blue box labeled "Vector o señal de largo N" has an arrow pointing to the summation index r .
- A blue box labeled "Transformada de Fourier" has an arrow pointing to the summation index r .
- A blue box containing the text "La transformada inversa de Fourier transforma una señal en la frecuencia a una en el espacio." is positioned to the right of the formula.

▶ Transformada IDFT

$$x_n = \frac{1}{N} \sum_{r=0}^{N-1} X_r \cdot e^{\frac{2\pi i}{N} \cdot r \cdot n}$$



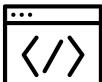
```
def ifft(X):
    N = len(X)

    if N <= 1:
        return X

    tmp = np.zeros(N, dtype=np.complex)
    x = np.zeros(N, dtype=np.complex)

    for n in range(0,N):
        for r in range(0,N):
            tmp[r] = X[r]*exp(2j*pi*r*n/N)
        x[n]=1/N* np.sum(tmp)
    return x
```

Ejercicio.



- ▶ Con los códigos anteriores compruebe qué sucede si ejecutamos el siguiente código.

```
A= [5.0, 4.0, 2.0, 1.0, 7.0, 9.0]
F= fft(A)
iF = ifft(F)

print(abs(iF))
```

Observe que la suma de los valores de cada matriz es cero.



```
def fft(x):
    N = len(x)

    if N <= 1:
        return x
    tmp = np.zeros(N, dtype=np.complex)
    X = np.zeros(N, dtype=np.complex)

    for r in range(0,N):
        for n in range(0,N):
            tmp[n]= x[n]*exp(-2j*pi*r*n/N)
        X[r]= np.sum(tmp)
    return X
```

```
def ifft(X):
    N = len(X)

    if N <= 1:
        return X

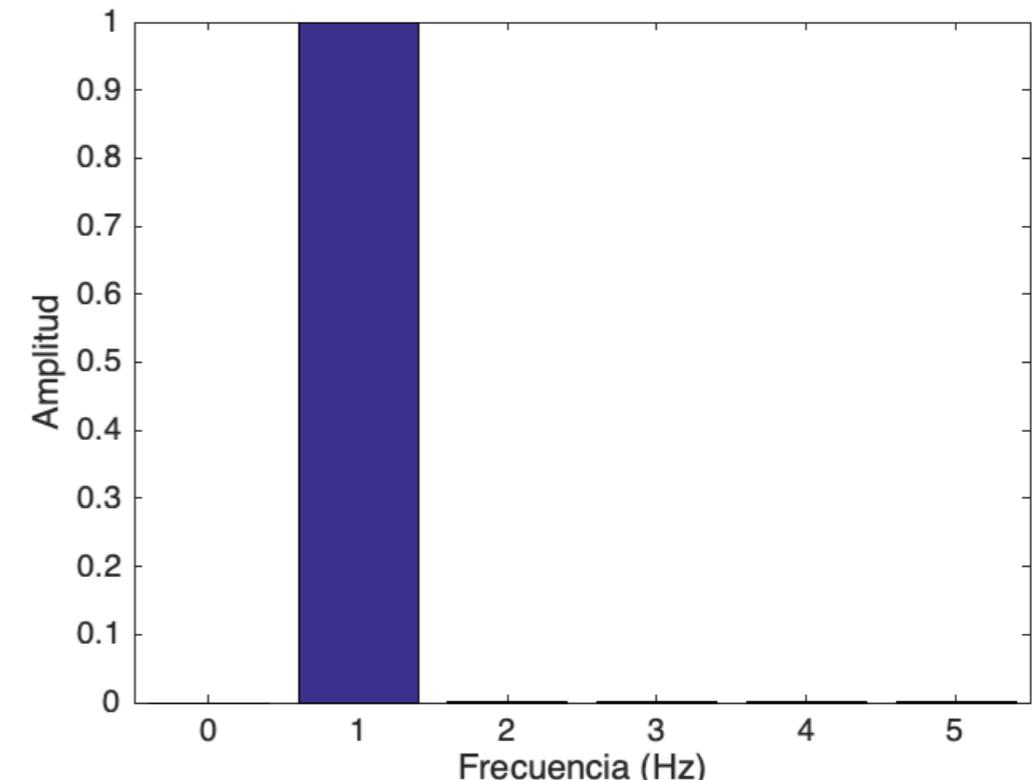
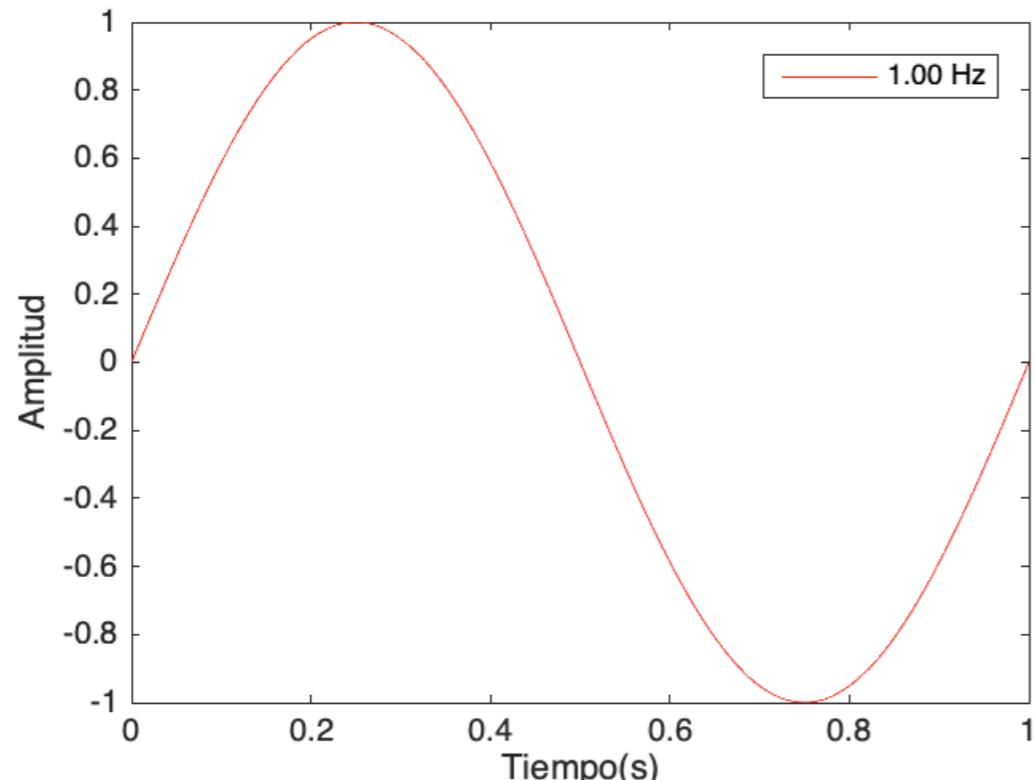
    tmp = np.zeros(N, dtype=np.complex)
    x = np.zeros(N, dtype=np.complex)

    for n in range(0,N):
        for r in range(0,N):
            tmp[r]= X[r]*exp(2j*pi*r*n/N)
        x[n]=1/N* np.sum(tmp)
    return x
```

- Mejoramiento en la frecuencia
 - Preliminar
 - Señales 1D
 - Señales 2D (imágenes)
 - Aplicaciones



- ▶ Recordemos algunos puntos de la señal periódica



Velocidad angular (rad/seg)

$$\omega = 2\pi f$$

Frecuencia (Hz)

Frecuencia (Hz)

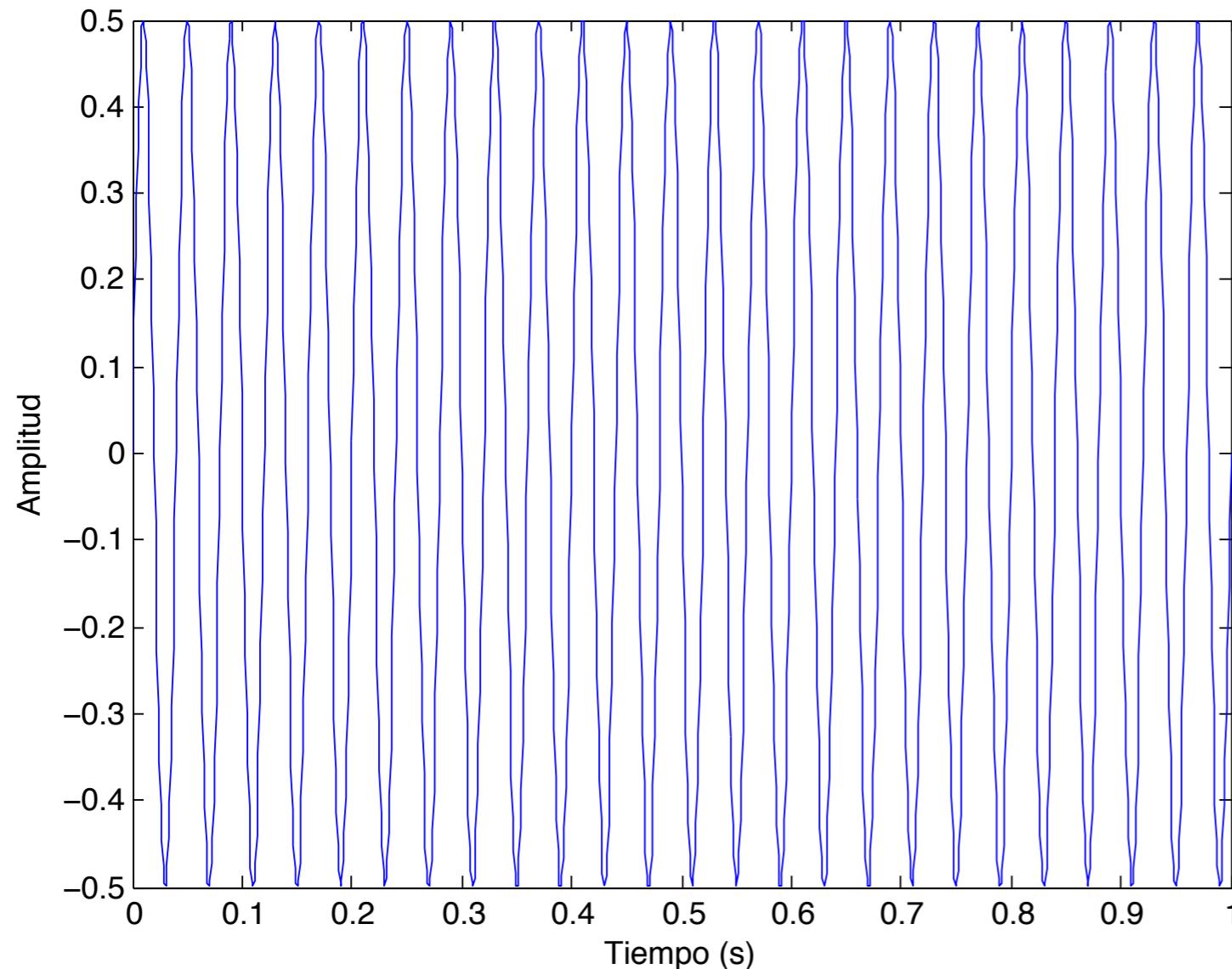
$$f = \frac{1}{T}$$

Tiempo



Ejemplo 1

- ▶ Suponga que ahora no sabes qué frecuencia tiene una señal.



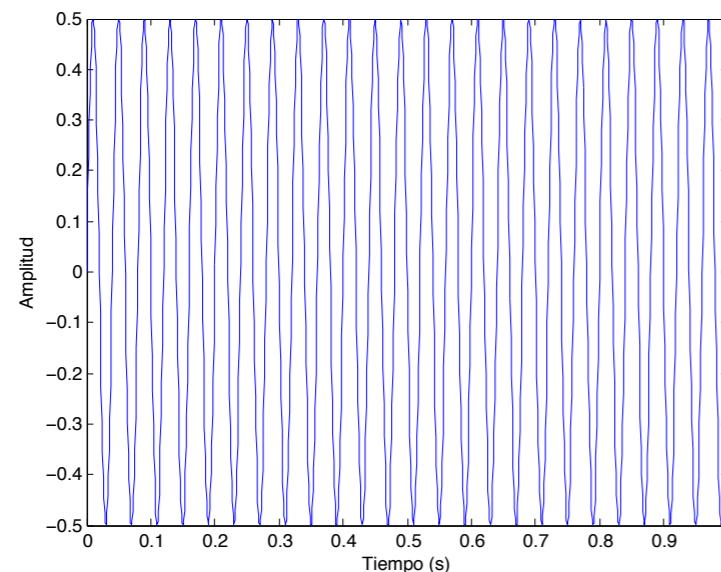
Qué frecuencia tiene la señal
que muestra la figura
(suponga que desconoce
cómo la determinamos)

¿cómo la
determinamos?



Ejemplo 1

- ▶ Primero: generemos el tiempo de la señal; en este ejemplo espaciada a un 1 milisegundo (eje x)



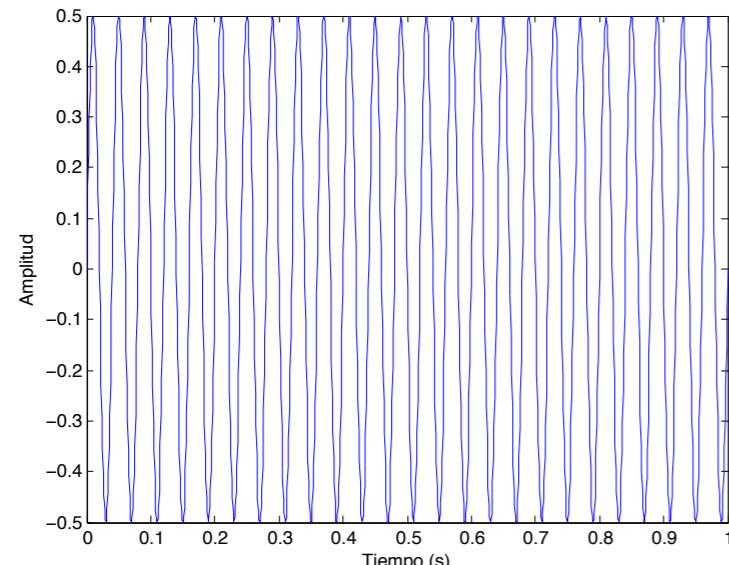
```
import numpy as np
import matplotlib.pyplot as plt

t= np.linspace(0,1, num=1000)
```

El comando **linspace**, genera una separación lineal entre dos valores separados por una cantidad fija de valores



- ▶ Segundo: generemos una señal de 25Hz y 0.5 de amplitud



Frecuencia $f = \frac{1}{T} = 25 \text{ Hz}$ $\rightarrow T = \frac{1}{f} = \frac{1}{25} = 0.04 \text{ seg}$

*o ciclos por segundo
se mide en Hertz*

*Significa 25 ciclos en
1 segundo*

$\omega = 2\pi f$ Esto es la velocidad angular, es decir,
ciclos por segundo. Se mide en rad/s

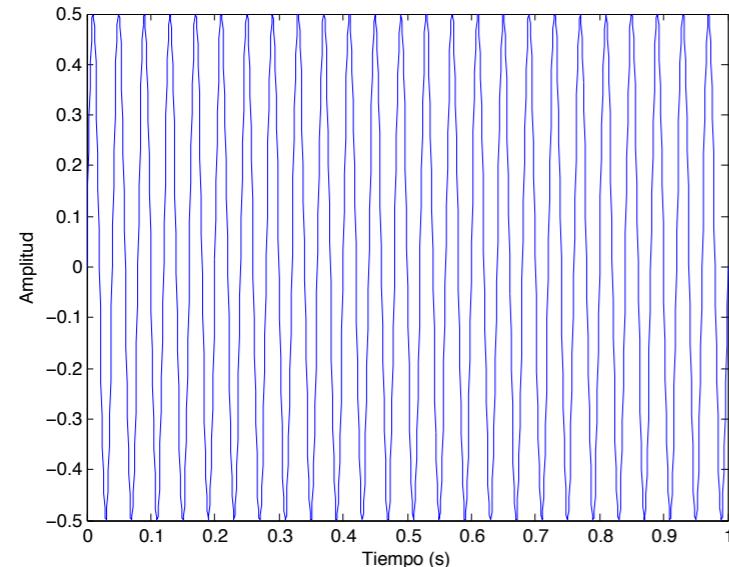
```
import numpy as np
import matplotlib.pyplot as plt

t= np.linspace(0,1, num=1000)
y = 0.5*np.sin(2*np.pi*t*25)
```

Inventemos una señal
sinusoidal de 25 Hz y 0.5
de amplitud

Esto significa que generamos
25 ciclos en un segundo con
una amplitud de 0.5.

- ▶ Tercero: generemos una señal de 25Hz y 0.5 de amplitud



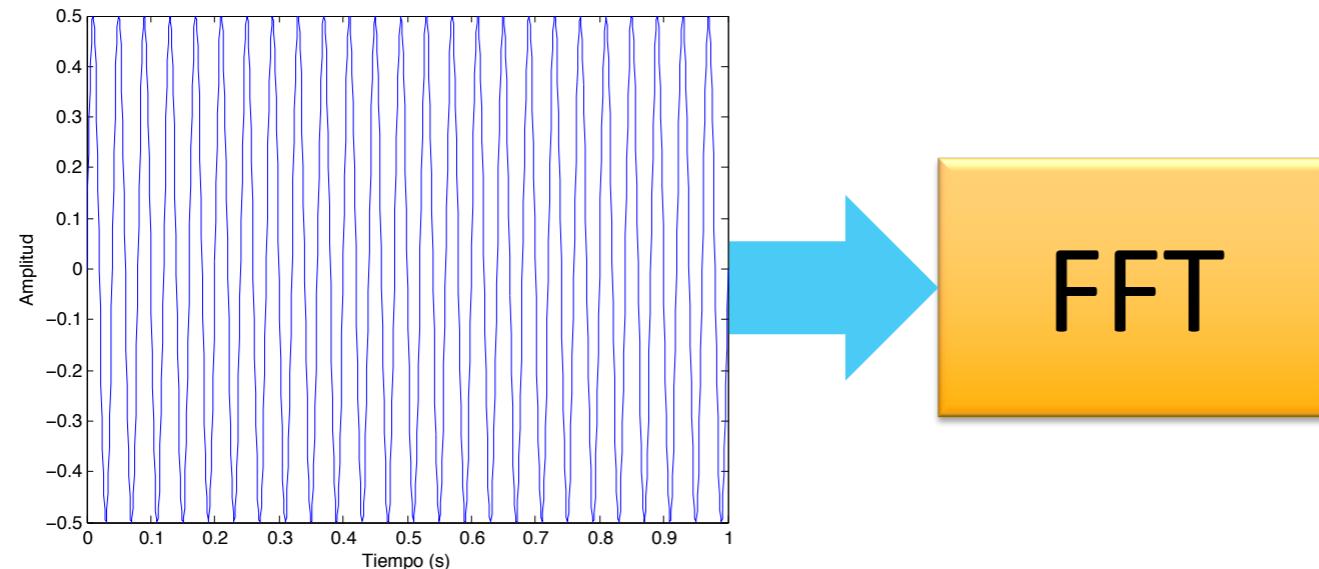
```
import numpy as np
import matplotlib.pyplot as plt

t= np.linspace(0,1, num=1000)
y = 0.5*np.sin(2*np.pi*t*25)

plt.plot(t, y)
plt.show()
```

Ploteamos la señal
(Tiempo vs Amplitud)

- ▶ Cuarto: Calculemos la Transformada de Fourier de la señal original.



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

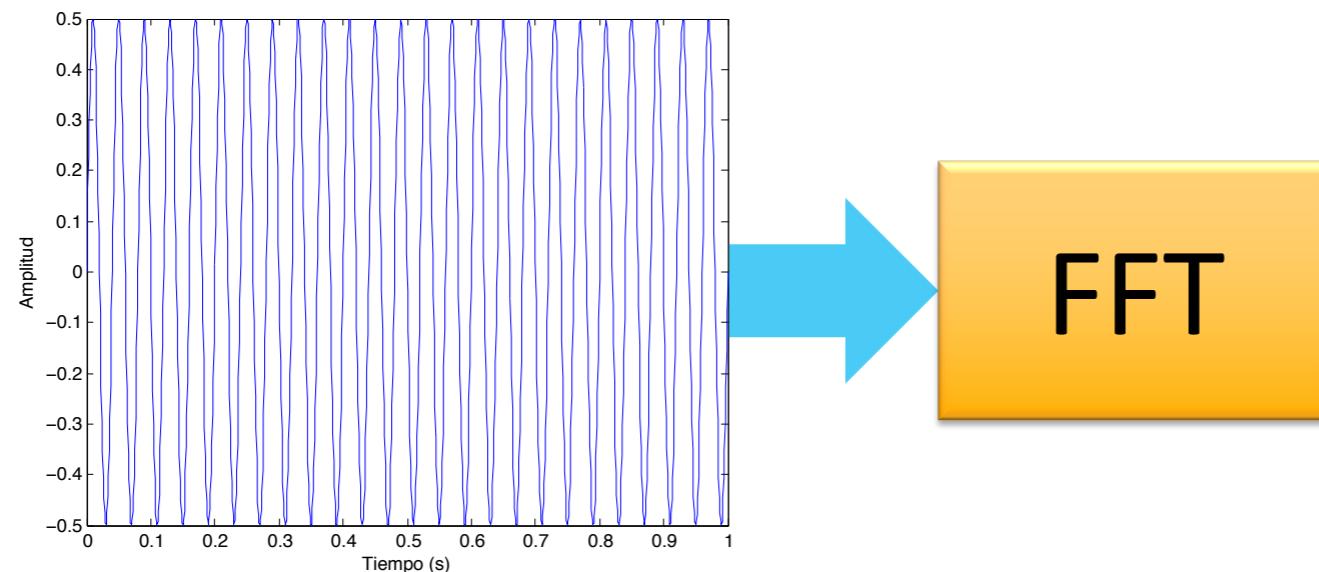
t= np.linspace(0,1, num=1000)
y = 0.5*np.sin(2*np.pi*t*25)

plt.plot(t, y)
plt.show()

F = np.fft.fft(y)
```

Aplicamos la Transformada
Discreta de Fourier (DFT)
(ultra rápida)

- ▶ Quinto: Ploteamos el tiempo versus el valor absoluto de la señal

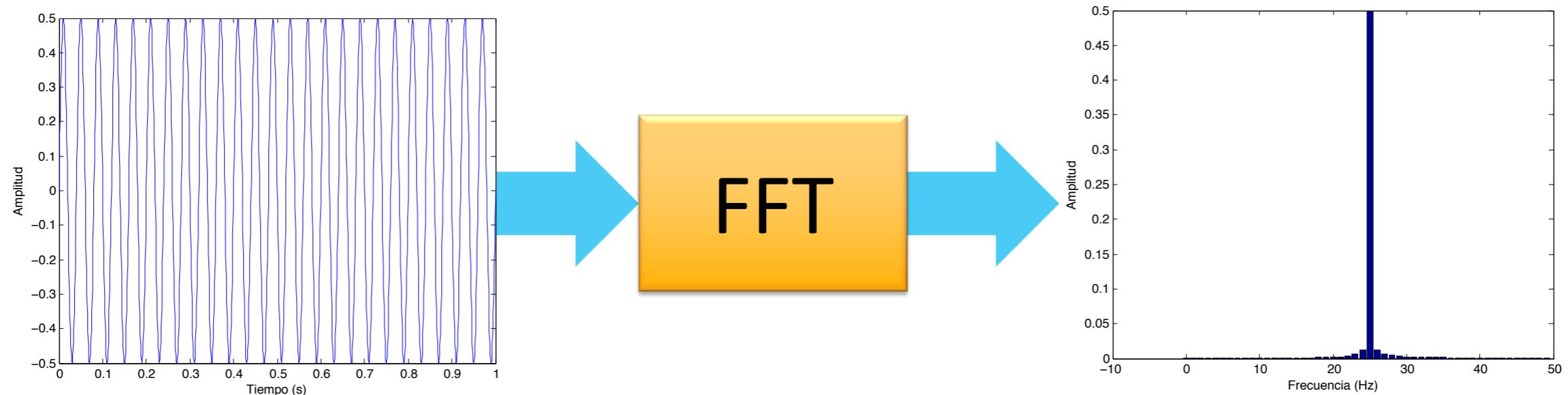


```
t= np.linspace(0,1, num=1000)
y = 0.5*np.sin(2*np.pi*t*25)

F = np.fft.fft(y)
N = len(F)
```

Calculamos el número de puntos de la muestra

- ▶ Quinto: Ploteamos el tiempo versus el valor absoluto de la señal



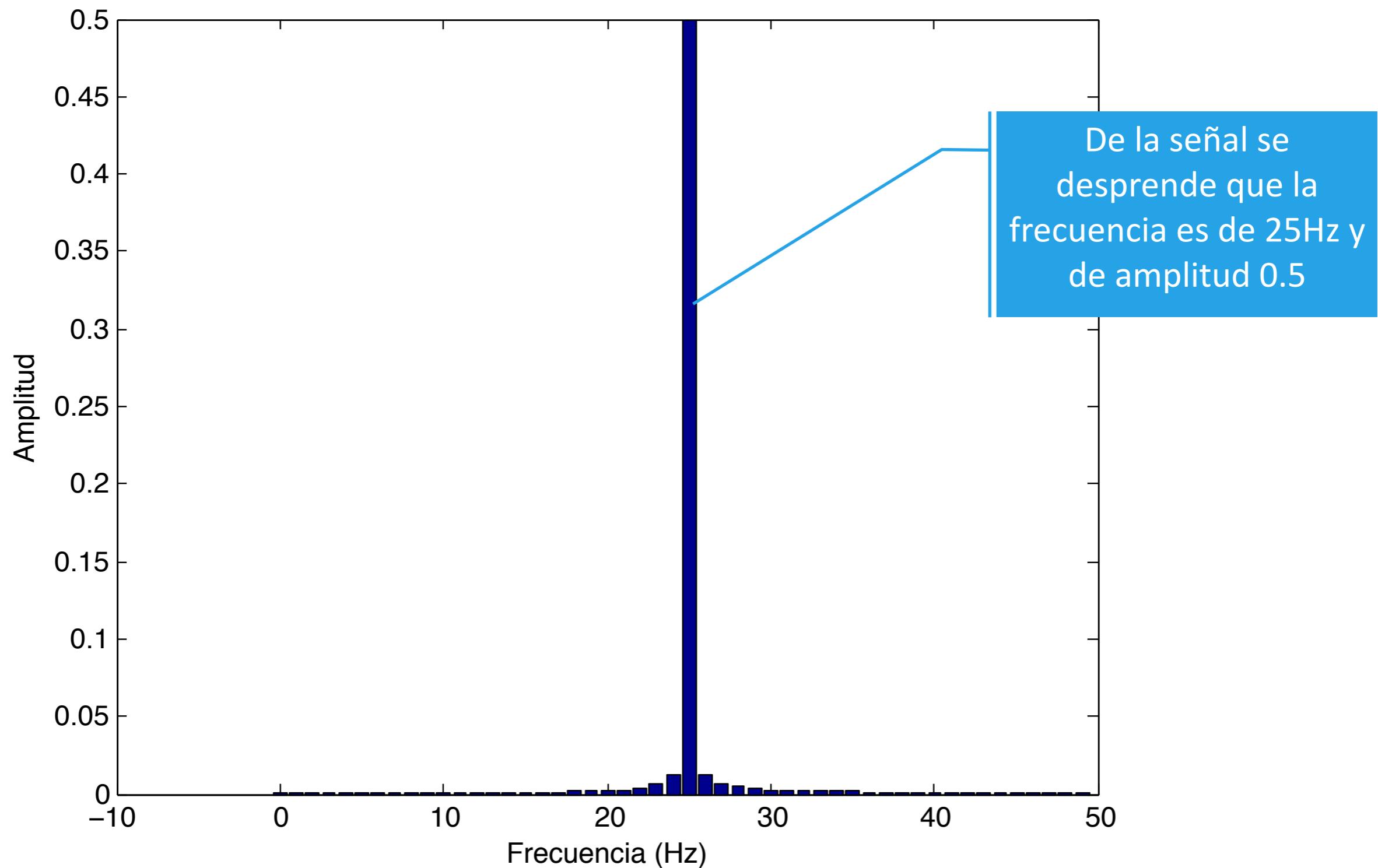
```
t= np.linspace(0,1, num=1000)
y = 0.5*np.sin(2*np.pi*t*25)

F = np.fft.fft(y)
N = len(F)

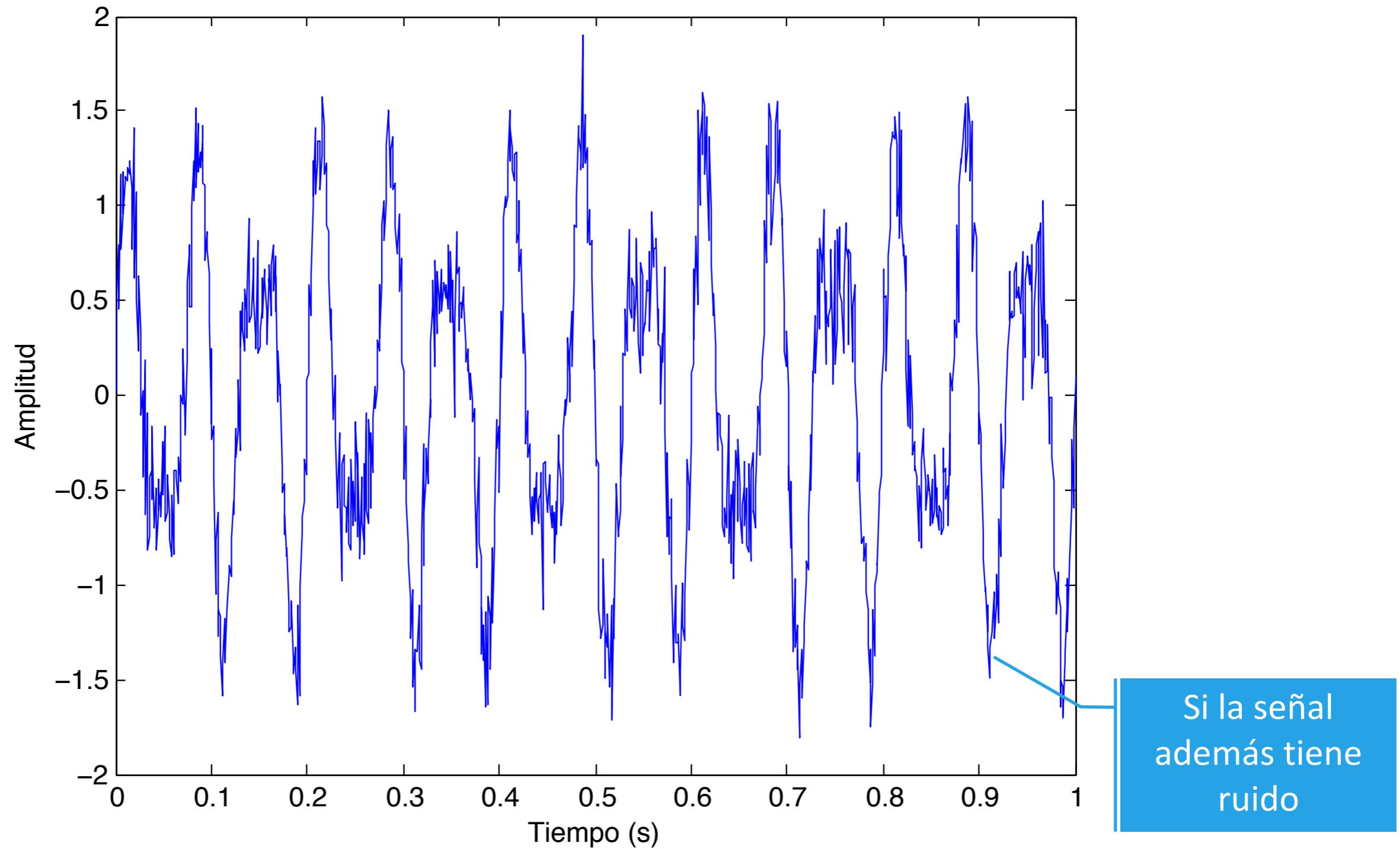
p= np.linspace(0, 49, dtype=int, num=50)
plt.bar(x=p, height=abs(F[p])/N*2)
plt.show()
```

Calculamos el valor absoluto de la señal para obtener la amplitud.
(en este caso desde 1 a 50Hz)

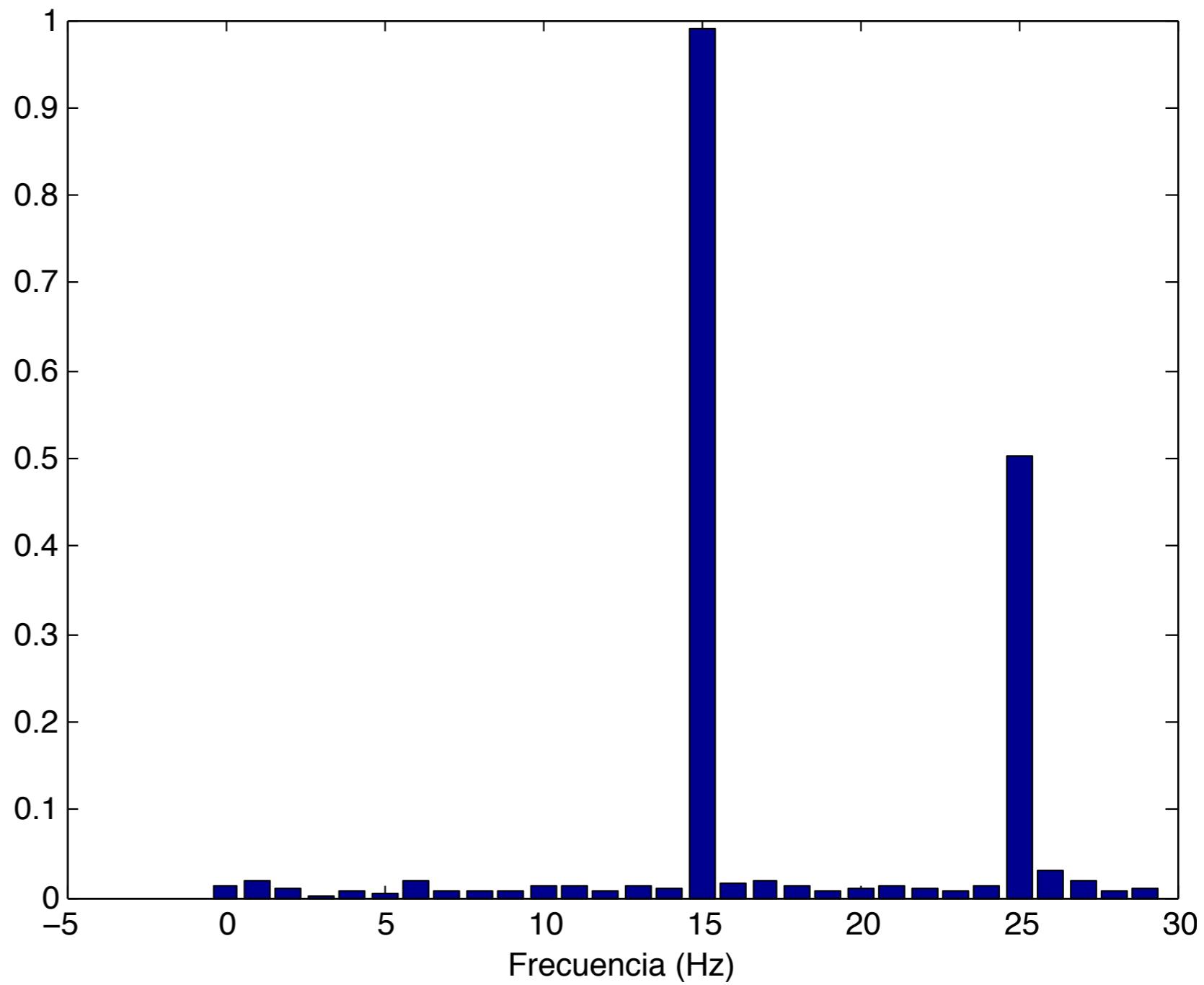
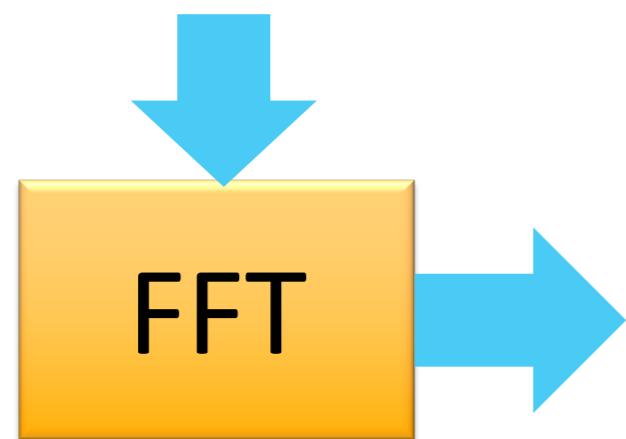
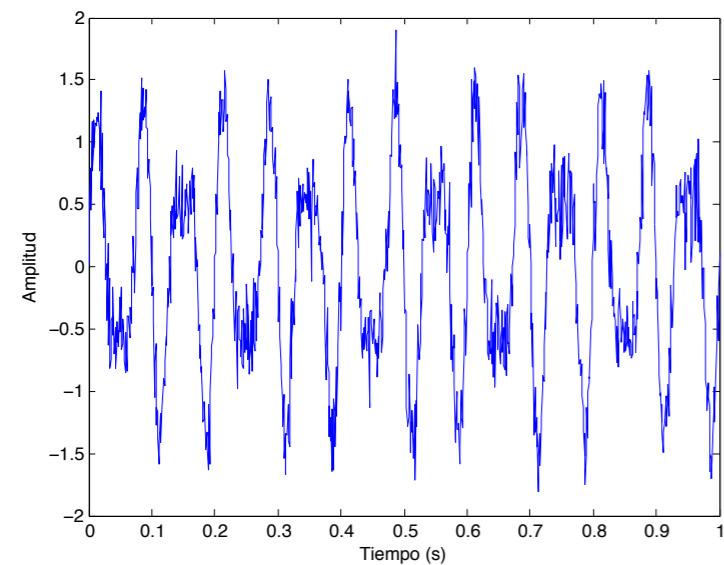
- ▶ Más en detalle



- ▶ Compliquemos las cosas, ¿qué frecuencia tiene la siguiente señal?

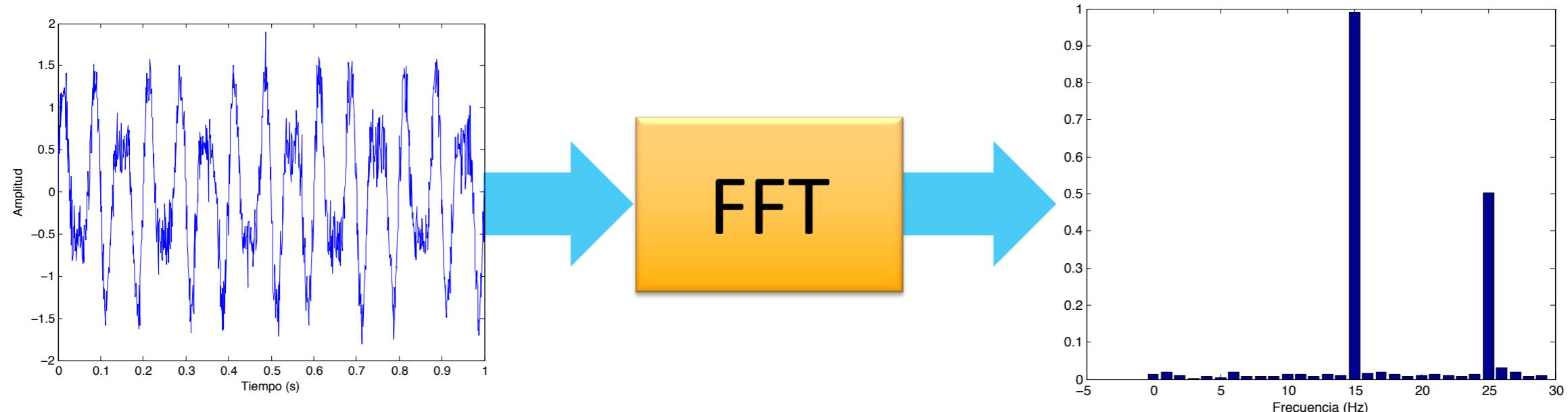


- ▶ Ahora es más claro que la señal es aproximadamente



Ejemplo 2

- La señal es aproximadamente: $0.5\sin(2\pi \cdot 25) + \sin(2\pi \cdot 15)$



```
t= np.linspace(0,1,num =1000) #tiempo
R = np.random.random(1000)

y = 0.5*np.sin(2*np.pi*25*t)+np.sin(2*np.pi*15*t)+0.2*R
```

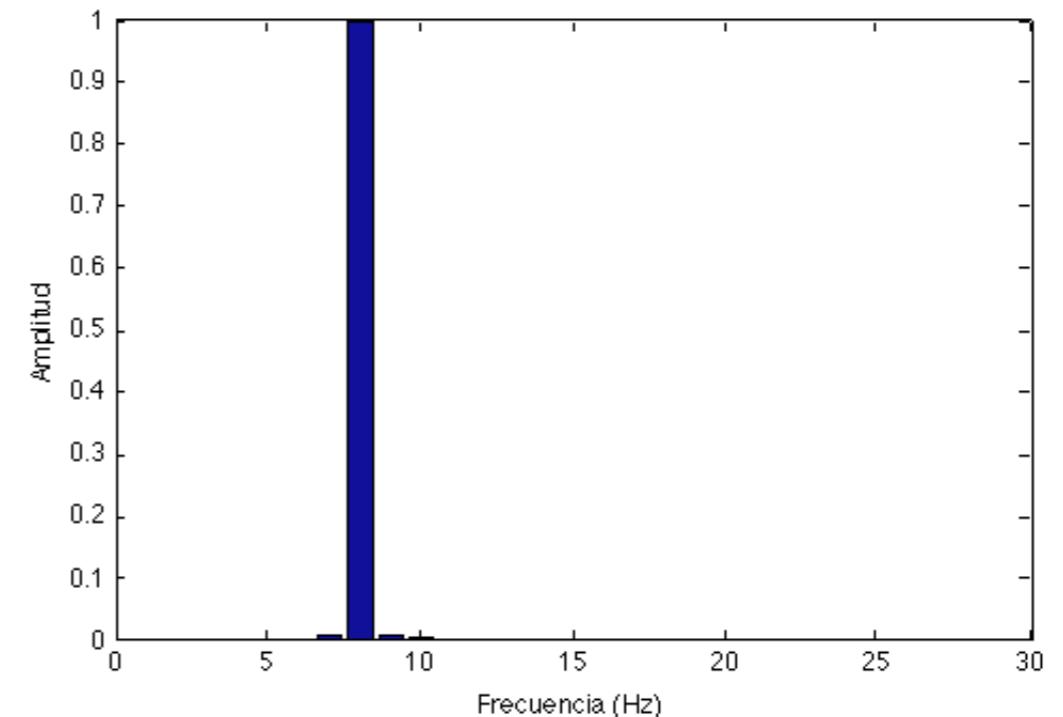
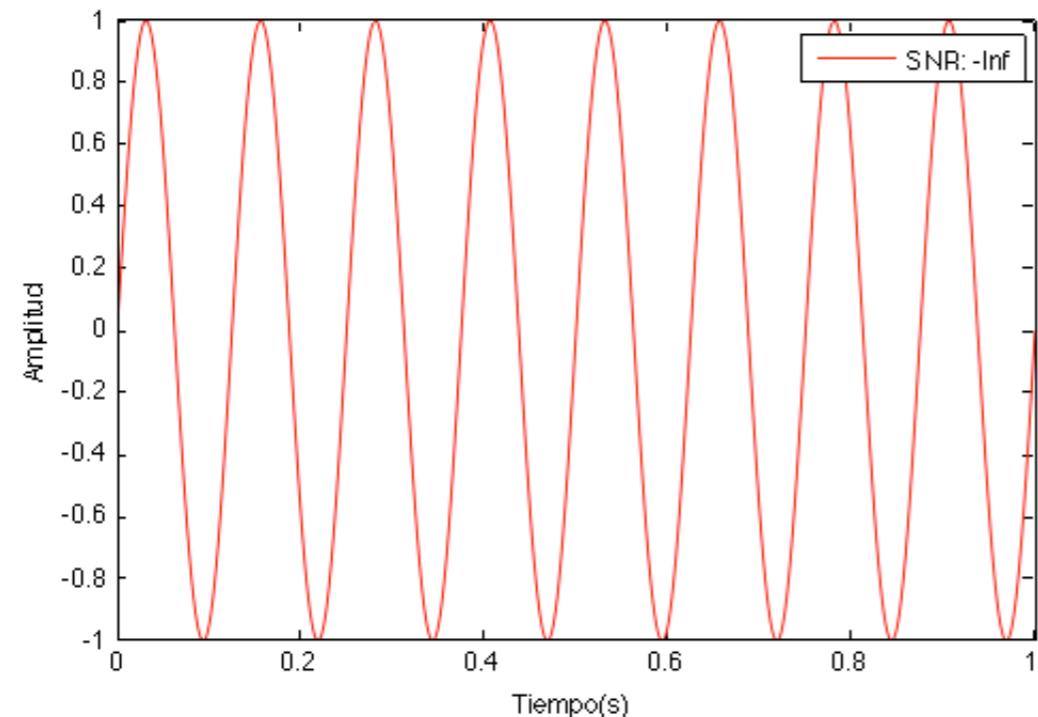
Aunque la señal tiene ruido, aún es posible encontrar las frecuencias correctas

```
F = np.fft.fft(y)
N = len(F)

p= np.linspace(1,49,dtype=int, num=50)
plt.bar(x=p, height=abs(F[p])/N*2)
plt.show()
```

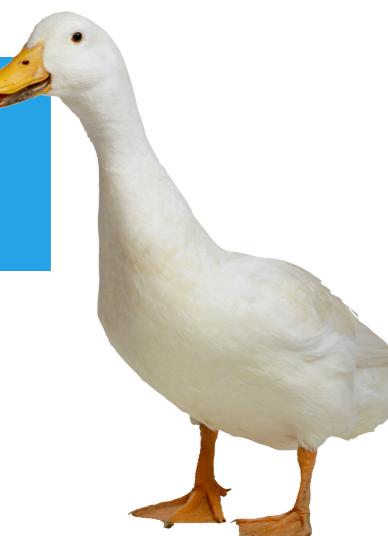
Buscando las señales

- ▶ Ejemplo de señal de 8 Hz con ruido gausiano normal



- Señal original: $\sin(2\pi \cdot 8)$

A pesar del ruido, aún es posible ver la señal que está de fondo.



- Mejoramiento en la frecuencia
 - Preliminar
 - Señales 1D
 - Señales 2D (imágenes)
 - Aplicaciones



Transformada DFT en dos dimensiones

- ▶ Transformación Discreta de Fourier en dos dimensiones (DFT2)

$$X_{r,s} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cdot e^{-i\frac{2\pi}{N} \cdot (s \cdot i + r \cdot j)}$$

Transformada
de Fourier en
la posición (r,s)

Valor de gris de la
posición (i,j) en
una imagen

Para $r = 0 \dots N - 1$
 $s = 0 \dots N - 1$

r: filas
s: columnas.



- ▶ Transformación Discreta de Fourier en dos dimensiones (DFT2)

$$X_{r,s} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cdot e^{-\iota \frac{2\pi}{N} \cdot (s \cdot i)} \cdot e^{-\iota \frac{2\pi}{N} \cdot (r \cdot j)}$$



$$X_{r,s} = \sum_{i=0}^{N-1} \left(\sum_{j=0}^{N-1} f(i,j) \cdot e^{-\iota \frac{2\pi}{N} \cdot (s \cdot j)} \right) \cdot e^{-\iota \frac{2\pi}{N} \cdot (r \cdot i)}$$

¿Qué expresión es esta
respecto a la DFT?

- ▶ Transformación Discreta de Fourier en dos dimensiones (DFT2)

$$X_{r,s} = \sum_{i=0}^{N-1} \left(\sum_{j=0}^{N-1} f(i,j) \cdot e^{-\frac{2\pi}{N} \cdot (s \cdot j)} \right) \cdot e^{-\frac{2\pi}{N} \cdot (r \cdot i)}$$

$$X_{r,s} = \sum_{i=0}^{N-1} X_s \cdot e^{-\frac{2\pi}{N} \cdot (r \cdot i)}$$

Esto es una matriz donde cada columna contiene la FFT de la matriz original.

Calculamos esto a lo largo de las columnas.

¿Qué expresión es esta respecto a la DFT?

- ▶ Transformación Discreta de Fourier en dos dimensiones (DFT)

$$X_{r,s} = \sum_{i=0}^{N-1} X_s \cdot e^{-\frac{2\pi}{N} \cdot (r \cdot i)}$$

Rotamos las columnas a filas y aplicamos nuevamente el algoritmo

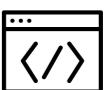
Se reduce a otra expresión?

$$X_{r,s} = X_r (X_s')'$$

Por lo tanto, aplicar una DFT en dos dimensiones simplemente significa aplicar en una DFT en las columnas de la imagen, y luego aplicar nuevamente otra DFT al resultado de ella.

Ejemplo en 2 dimensiones

- Veamos lo anterior en un ejemplo



A =

5	9	10
8	6	9
8	0	2

```
A= np.array([[ 5.0, 9.0, 10.0],  
             [ 8.0, 6.0, 9.0],  
             [ 8.0, 0.0, 2.0]])
```

```
print(fft(A[:,0]))
```

```
print(fft(A[:,1]))
```

```
print(fft(A[:,2]))
```

```
21.0000  
-3.0000 - 0.0000i  
-3.0000 - 0.0000i
```

```
15.0000  
6.0000 - 5.1962i  
6.0000 + 5.1962i
```

```
21.0000  
4.5000 - 6.0622i  
4.5000 + 6.0622i
```

Ejemplo en 2 dimensiones

- ▶ Veamos lo anterior en un ejemplo



A =

5	9	10
8	6	9
8	0	2

```
A= np.array([[ 5.0,  9.0, 10.0],  
             [ 8.0,  6.0,  9.0],  
             [ 8.0,  0.0,  2.0]])  
  
F= np.zeros([3,3], dtype=complex)  
for i in range(0,3):  
    F[:,i]= fft(A[:,i])  
  
print(F)
```

output

```
[[21. +0.0000000e+00j 15. +0.0000000e+00j 21.+0.0000000e+00j]  
[-3. -3.55271368e-15j 6. -5.19615242e+00j 4.5-6.06217783e+00j]  
[-3. -6.21724894e-15j 6. +5.19615242e+00j 4.5+6.06217783e+00j]]
```

Ejemplo en 2 dimensiones

- ▶ Veamos lo anterior en un ejemplo

F

```
[ [21. +0.0000000e+00j 15. +0.0000000e+00j 21. +0.0000000e+00j ]
```

```
[ -3. -3.55271368e-15j 6. -5.19615242e+00j 4.5-6.06217783e+00j ]
```

```
[ -3. -6.21724894e-15j 6. +5.19615242e+00j 4.5+6.06217783e+00j ] ]
```

```
print(fft(F[0,:]))
```

```
print(fft(F[1,:]))
```

```
print(fft(F[2,:]))
```

```
57.0000  
3.0000 + 5.1962i  
3.0000 - 5.1962i
```

```
7.5000 -11.2583i  
-7.5000 + 4.3301i  
-9.0000 + 6.9282i
```

```
7.5000 +11.2583i  
-9.0000 - 6.9282i  
-7.5000 - 4.3301i
```

Ejemplo en 2 dimensiones

- ▶ Veamos lo anterior en un ejemplo

F =

$$\begin{array}{ccc} 21.0000 & 15.0000 & 21.0000 \\ -3.0000 - 0.0000i & 6.0000 - 5.1962i & 4.5000 - 6.0622i \\ -3.0000 - 0.0000i & 6.0000 + 5.1962i & 4.5000 + 6.0622i \end{array}$$

FF =

$$\begin{array}{ccc} 57.0000 & 3.0000 + 5.1962i & 3.0000 - 5.1962i \\ 7.5000 - 11.2583i & -7.5000 + 4.3301i & -9.0000 + 6.9282i \\ 7.5000 + 11.2583i & -9.0000 - 6.9282i & -7.5000 - 4.3301i \end{array}$$

Transformada de Fourier Doble

```
for i in range(0,3):
    FF[i,:]= fft(F[i,:])
```

Esto hace que el resultado quede en columnas



Ejemplo en 2 dimensiones

- ▶ Veamos lo anterior en un ejemplo

A=

5	9	10
8	6	9
8	0	2

Resumen

- fft() → Transformada unidimensional FFT
- ifft() → Inversa de Fourier
- fft2() → Transformada en 2D
- ifft2() → Inversa de transformada 2D

```
F = np.fft.fft2(A)
```

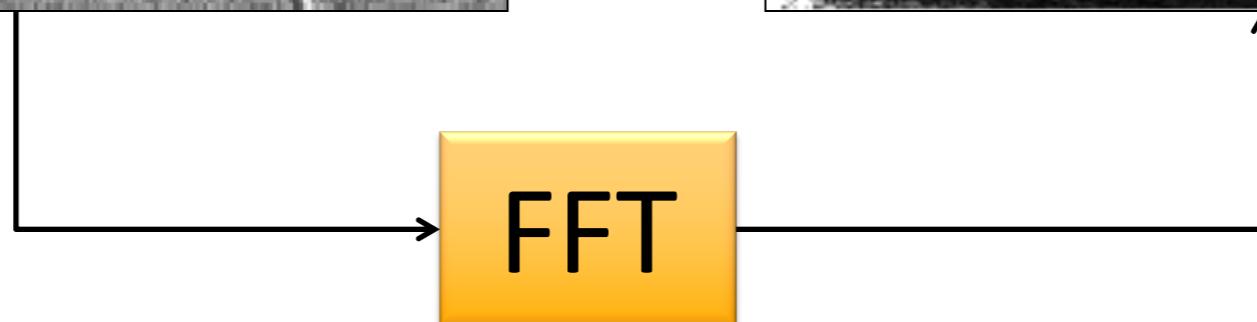
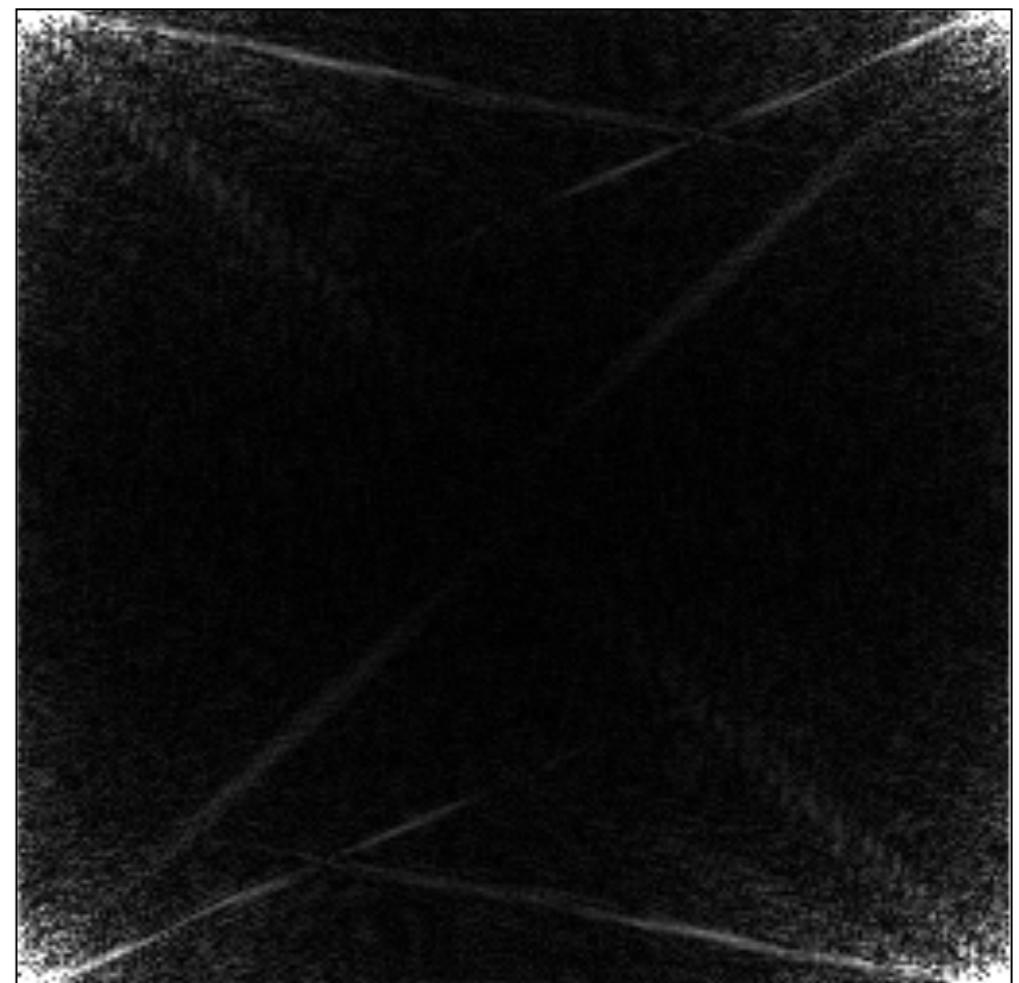
57.0000	3.0000 + 5.1962i	3.0000 - 5.1962i
7.5000 -11.2583i	-7.5000 + 4.3301i	-9.0000 + 6.9282i
7.5000 +11.2583i	-9.0000 - 6.9282i	-7.5000 - 4.3301i

Ejemplo en imágenes

- ▶ Al aplicar FFT a la imagen obtenemos un espectro



Espectro de Fourier



- ▶ El código que hace lo anterior es el siguiente:



```
import cv2
import numpy as np

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

Cuando leemos una imagen,
esta viene con tres canales.
por ello debemos emplear
solo uno, o bien
transformarla a niveles de
gris.



- ▶ El código que hace lo anterior es el siguiente:



```
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))
```

Dado que los valores son complejos, calculamos el valor absoluto, y luego el logaritmo



- ▶ El código que hace lo anterior es el siguiente:



```
import cv2
import numpy as np

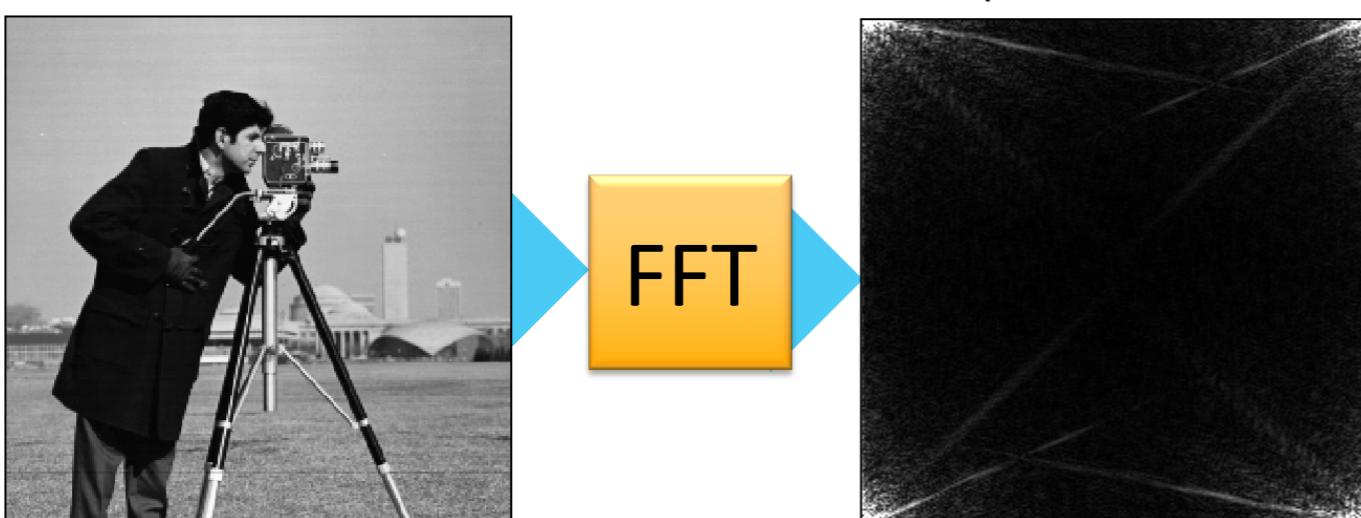
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

F = np.fft.fft2(gray)

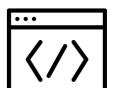
spectrum = np.log(np.abs(F))
out = cv2.normalize(spectrum, None, 0.0, 1.0, cv2.NORM_MINMAX)

cv2.imshow('fft2', out)
cv2.waitKey(0)
```

Espectro de Fourier



- ▶ Mejoremos la visualización del espectro



```
import cv2
import numpy as np

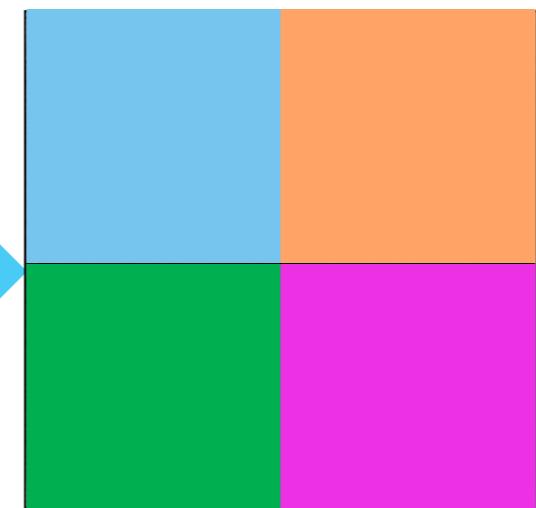
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

F = np.fft.fft2(gray)
fshift = np.fft.fftshift(F)
spectrum = np.log(np.abs(fshift))
out = cv2.normalize(spectrum, None, 0.0, 1.0, cv2.NORM_MINMAX)

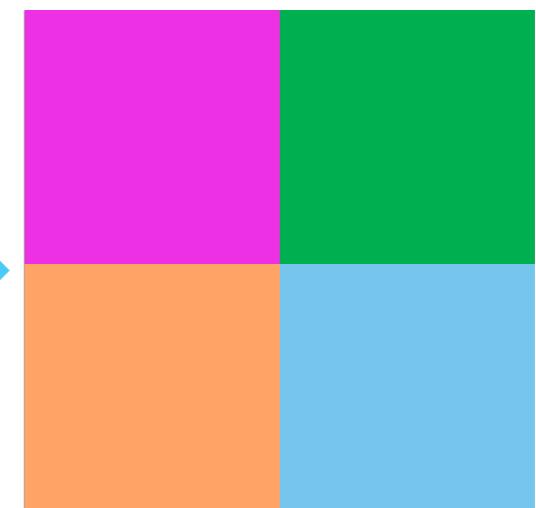
cv2.imshow('fft2', out)
cv2.waitKey(0)
```



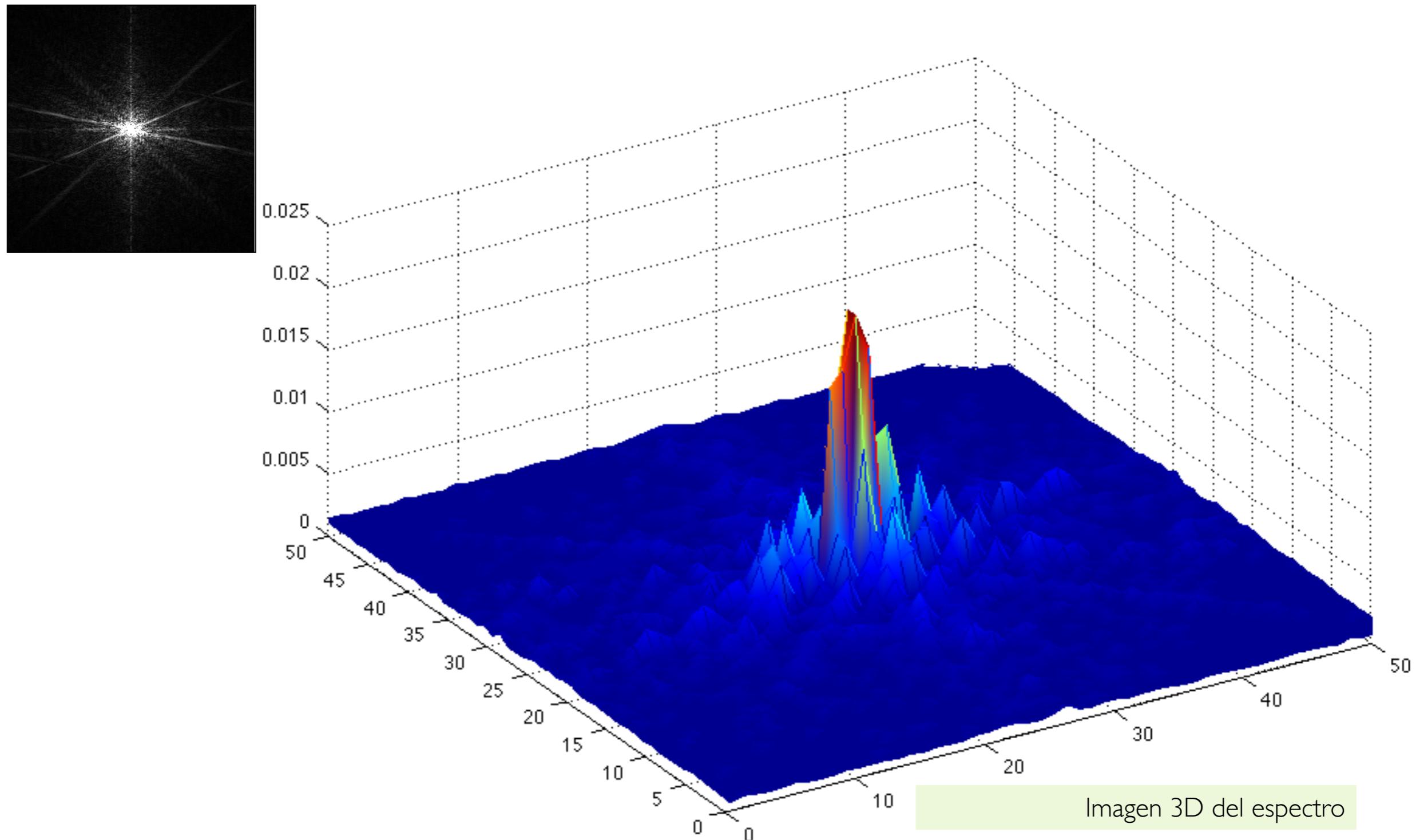
FFT



shift

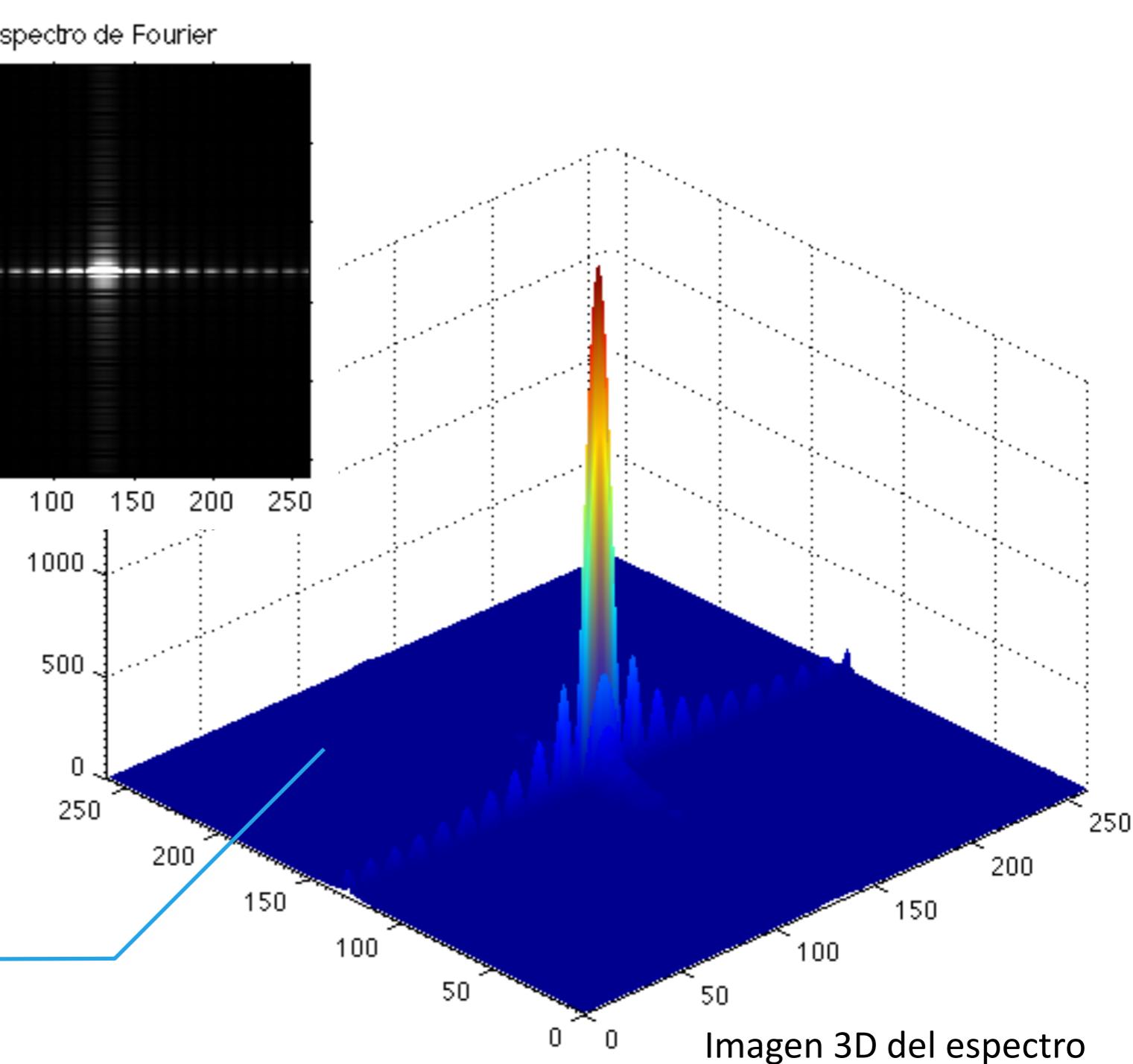
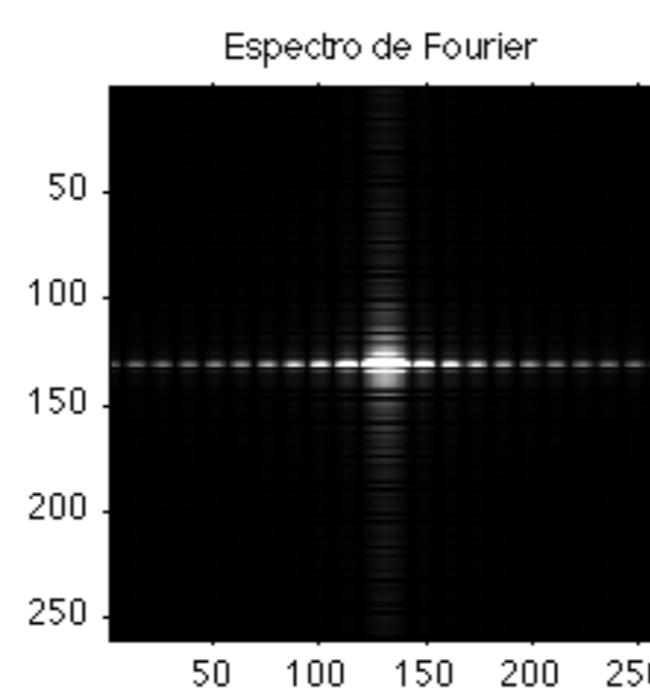
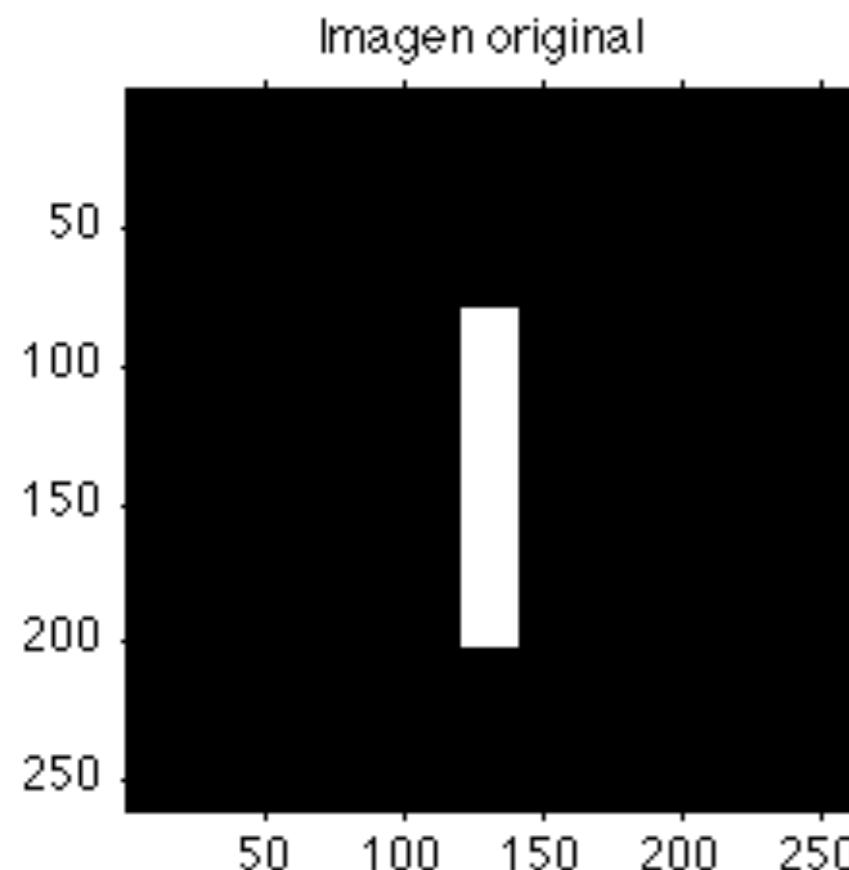


- ▶ Mejoremos la visualización del espectro



¿Qué muestra el espectro de Fourier?

- Veamos un ejemplo con imágenes muy simples

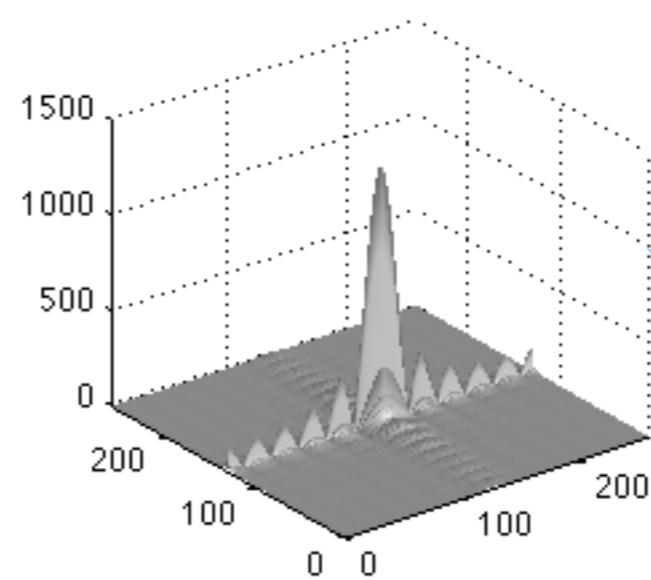
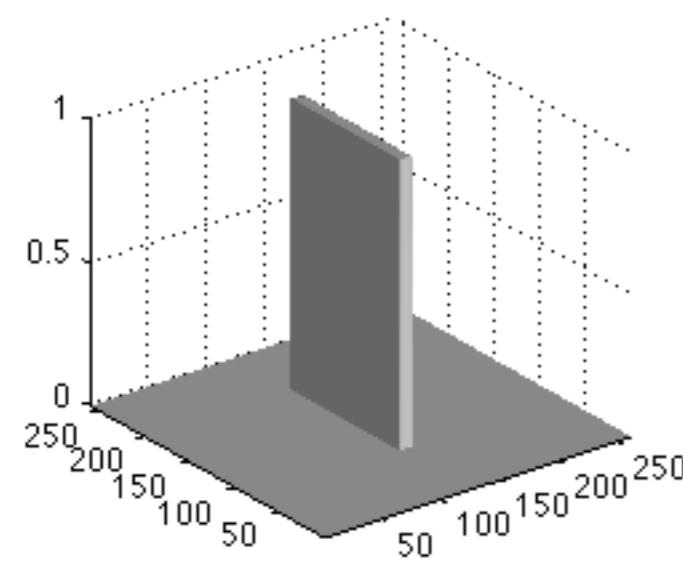
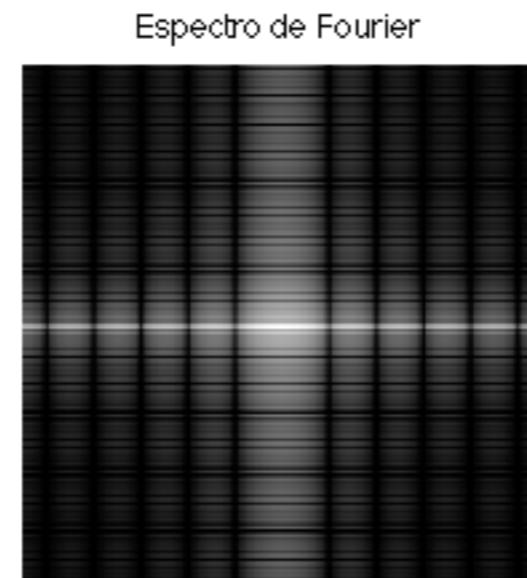
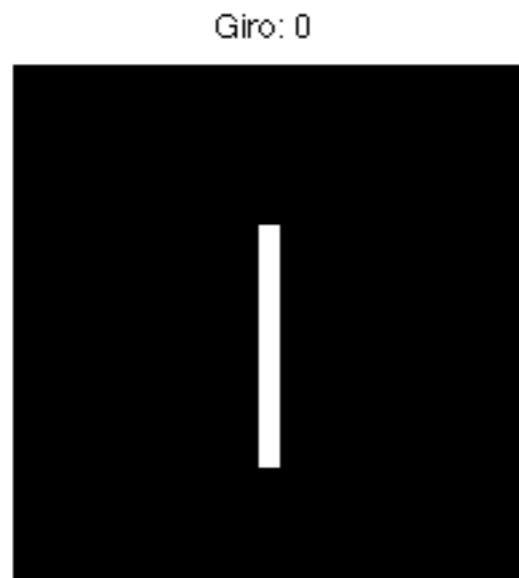


El espectro muestra la descomposición de la señal en componentes sinusoidales



¿Qué muestra el espectro de fourier?

- ▶ Veamos un ejemplo con imágenes muy simples



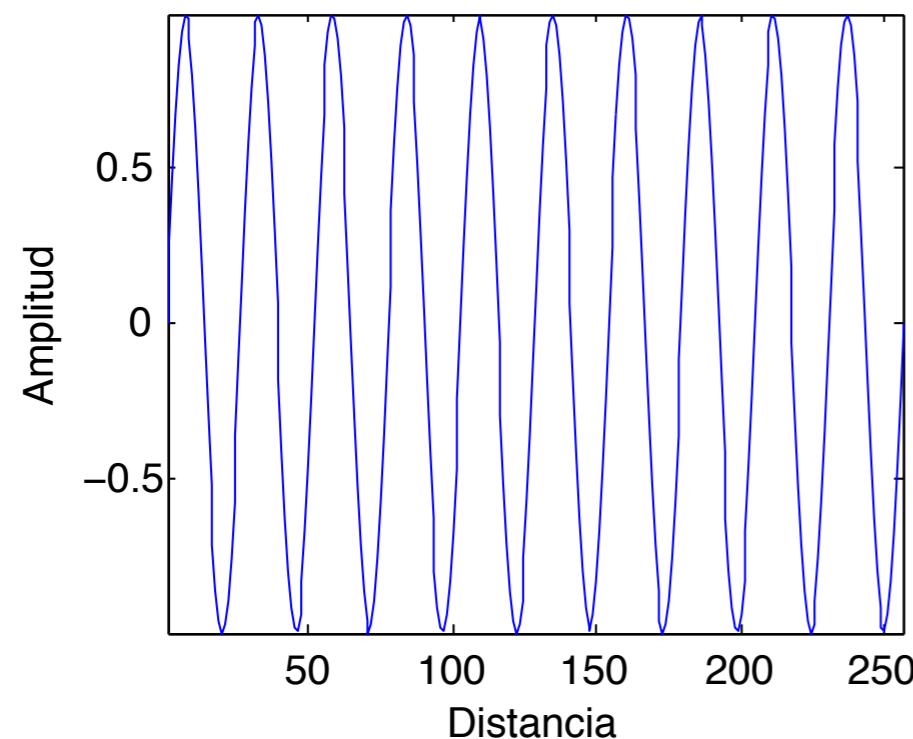
Note las frecuencias que se generan al girar la imagen.



- Mejoramiento en la frecuencia
 - Preliminar
 - Señales 1D
 - Señales 2D (imágenes)
 - Aplicaciones



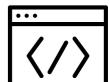
- ▶ Primer paso: Creemos una señal de ruido



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

V = np.linspace(0,1,num=256)
Y= 0.6*np.sin(2*np.pi*10*V)

plt.plot(V,Y)
plt.show()
```



- ▶ Primer paso: ¿Cuál es la DFT de la señal de ruido?

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

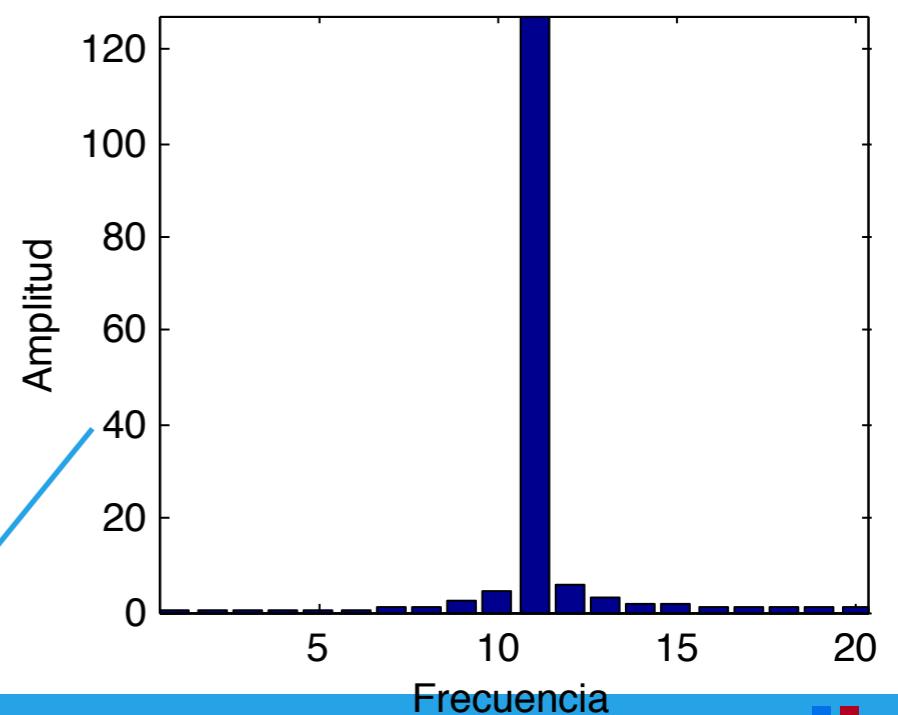
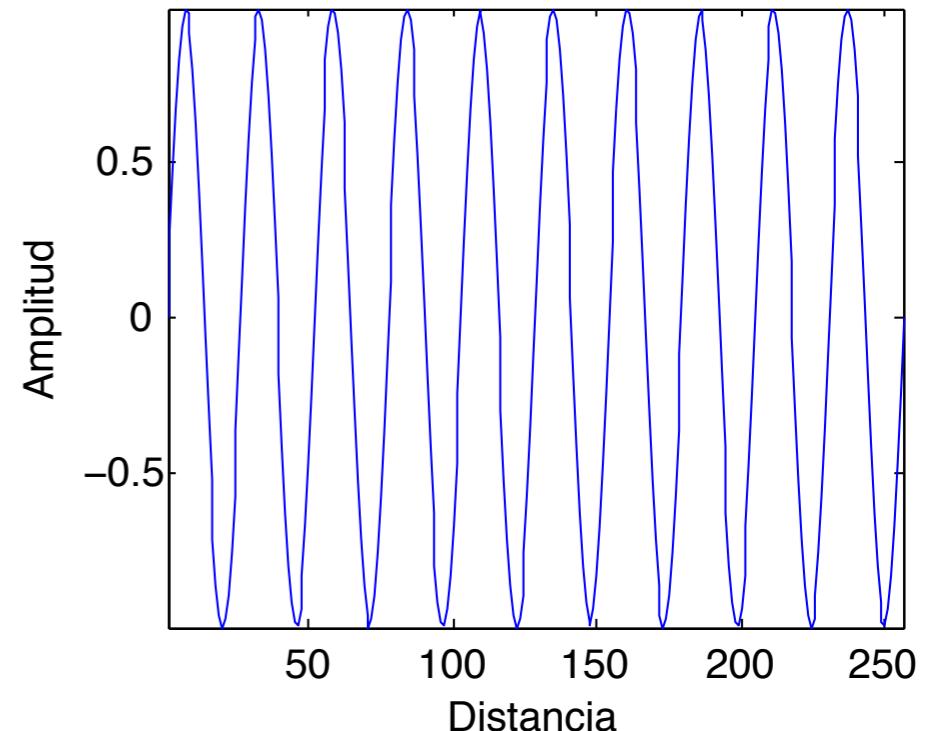
V = np.linspace(0,1,num=256)
Y= 0.6*np.sin(2*np.pi*10*V)

freq= np.fft.fft(Y)

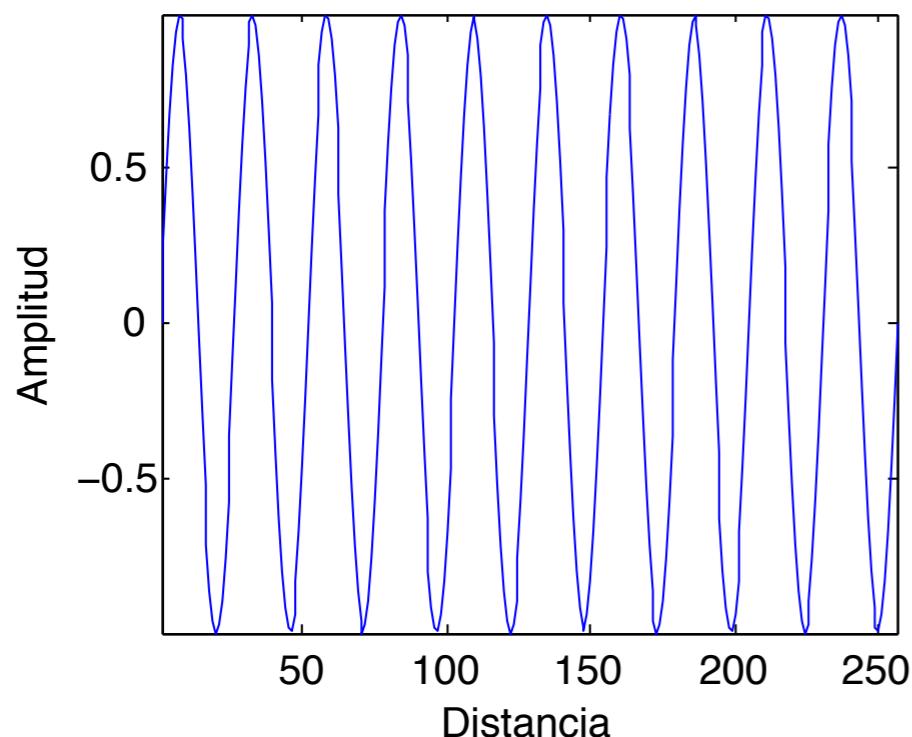
N= len(freq)
p= np.linspace(0,49,dtype=int, num=50)

plt.bar(x=p, height=abs(freq[p])/N*2)
plt.show()
```

FFT de la señal



- ▶ Segundo paso: Generar una imagen a partir de la señal



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

V = np.linspace(0,1,num=256)
Y= 0.6*np.sin(2*np.pi*10*V)

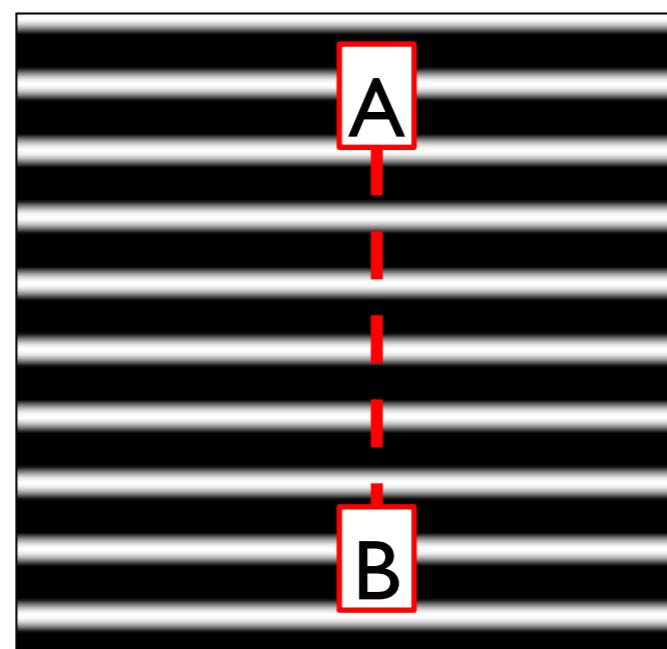
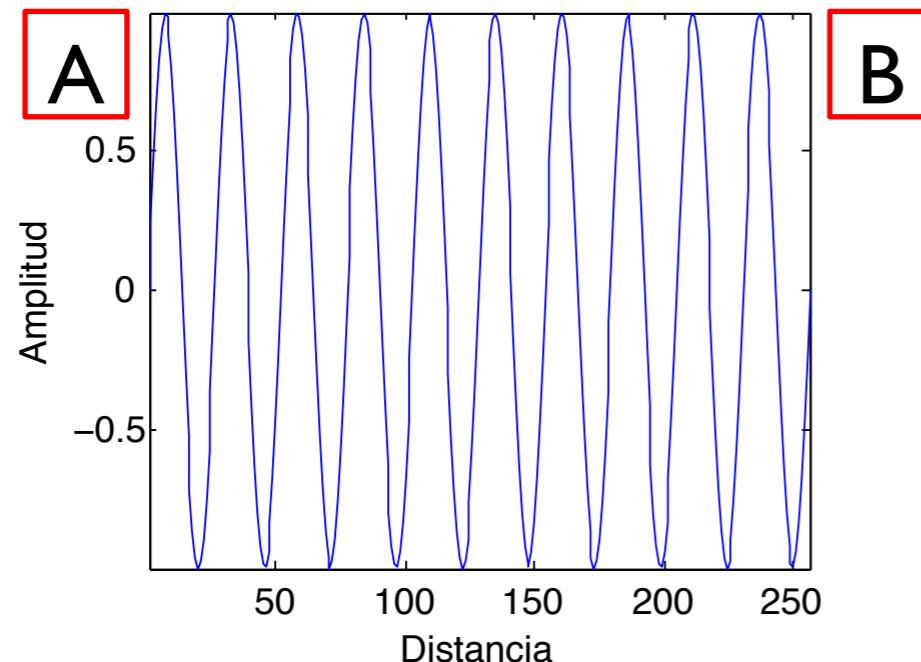
M= np.matlib.repmat(Y,256,1)
M= M.transpose()
```

```
a=[1 ,2 ,3]
b= np.matlib.repmat(a,3,1)
```

```
b =
1 2 3
1 2 3
1 2 3
```

La función
repmat repite el
vector n filas y m
columnas

- ▶ Cómo calculamos la imagen a partir de la señal



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

V = np.linspace(0,1,num=256)
Y= 0.6*np.sin(2*np.pi*10*V)

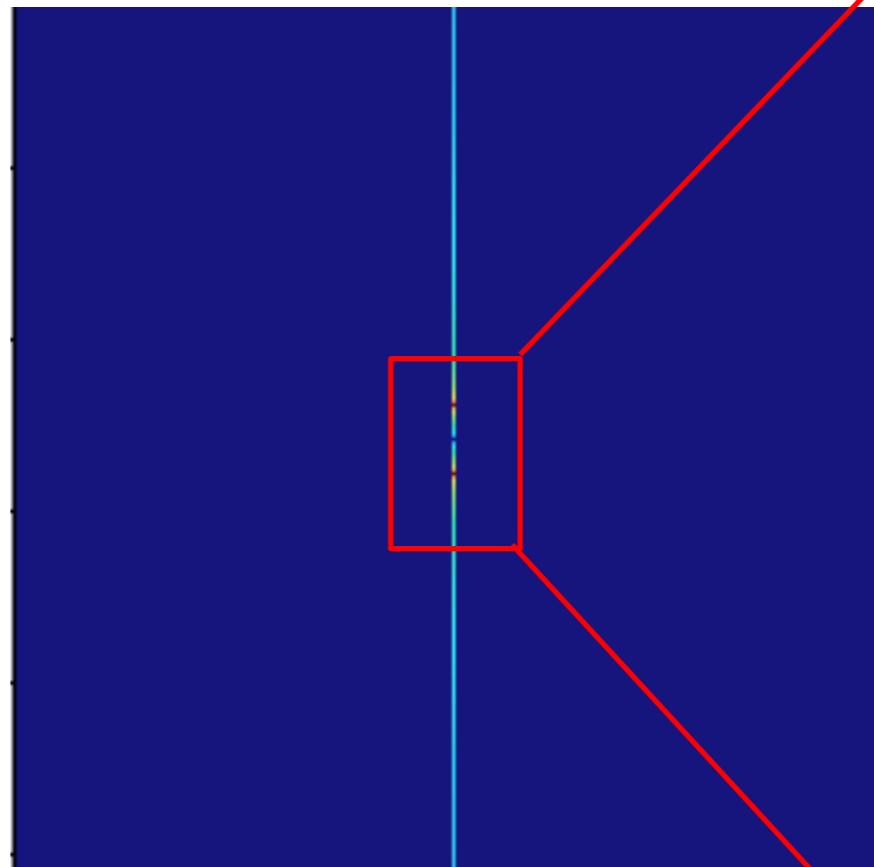
M= np.matlib.repmat(Y,256,1)
M= M.transpose()

cv2.imshow('signal', M)
cv2.waitKey(0)
```

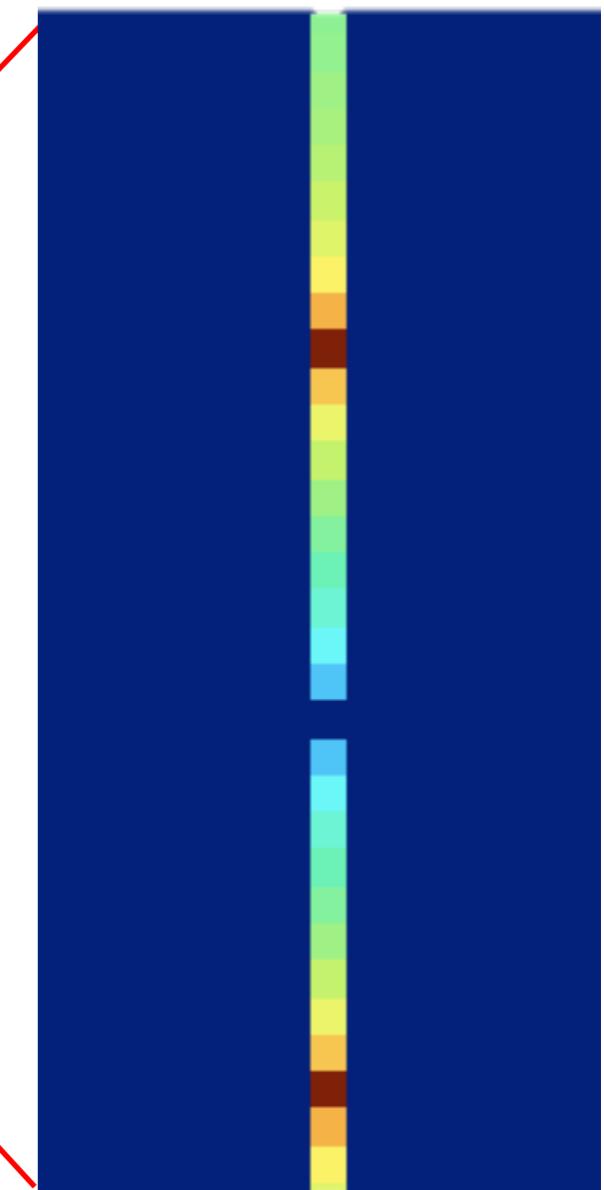
- ▶ Ejemplo de una aplicación de filtrado



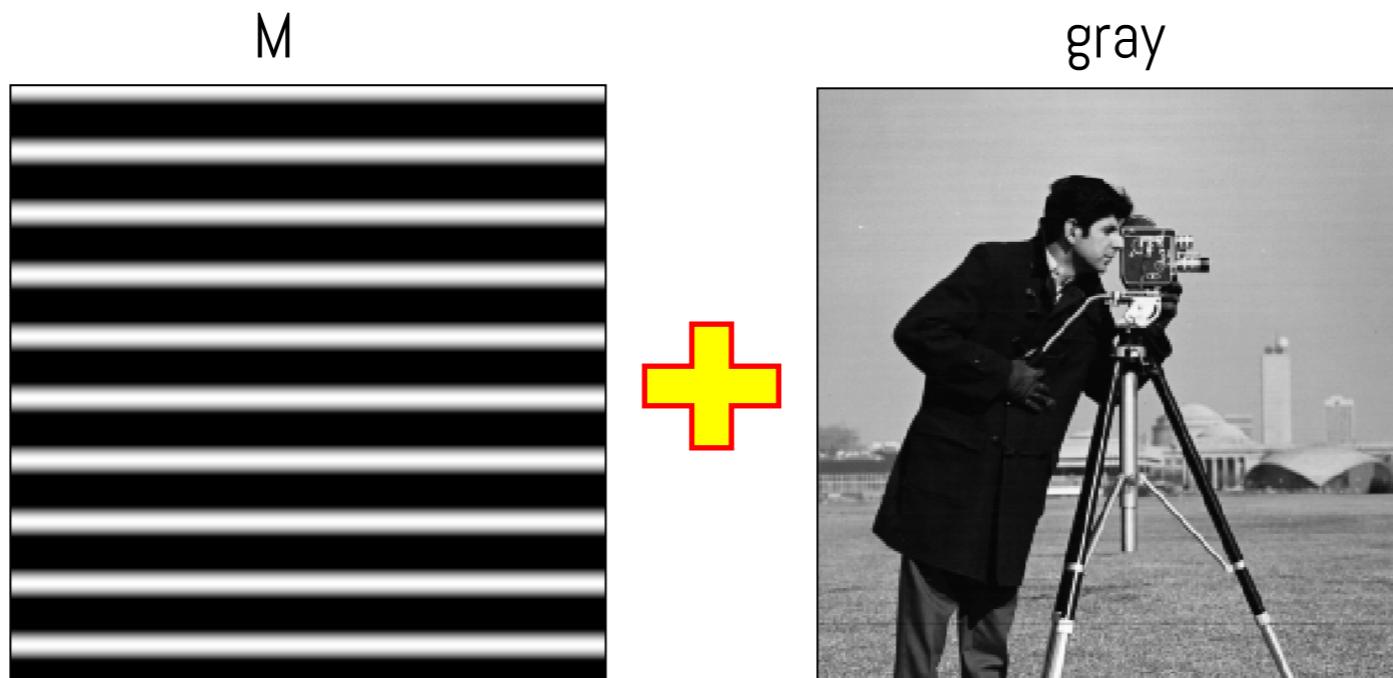
Imagen de ruido



Espectro



- ▶ Ejemplo de una aplicación de filtrado



```
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.normalize(gray.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)

V = np.linspace(0,1,num=256)
Y= 0.2*np.sin(2*np.pi*10*V)

M = np.matlib.repmat(Y,256,1)
M = M.transpose()

noise_img= np.add(M, gray)
```

La función add suma dos imágenes

- ▶ Ejemplo de una aplicación de filtrado



```
img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.normalize(gray.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)

V = np.linspace(0,1,num=256)
Y= 0.2*np.sin(2*np.pi*10*V)

M = np.matlib.repmat(Y,256,1)
M = M.transpose()

noise_img= np.add(M, gray)
```

- ▶ Calculemos el espectro de Fourier de la imagen con ruido, ¿qué notamos?

noise_img

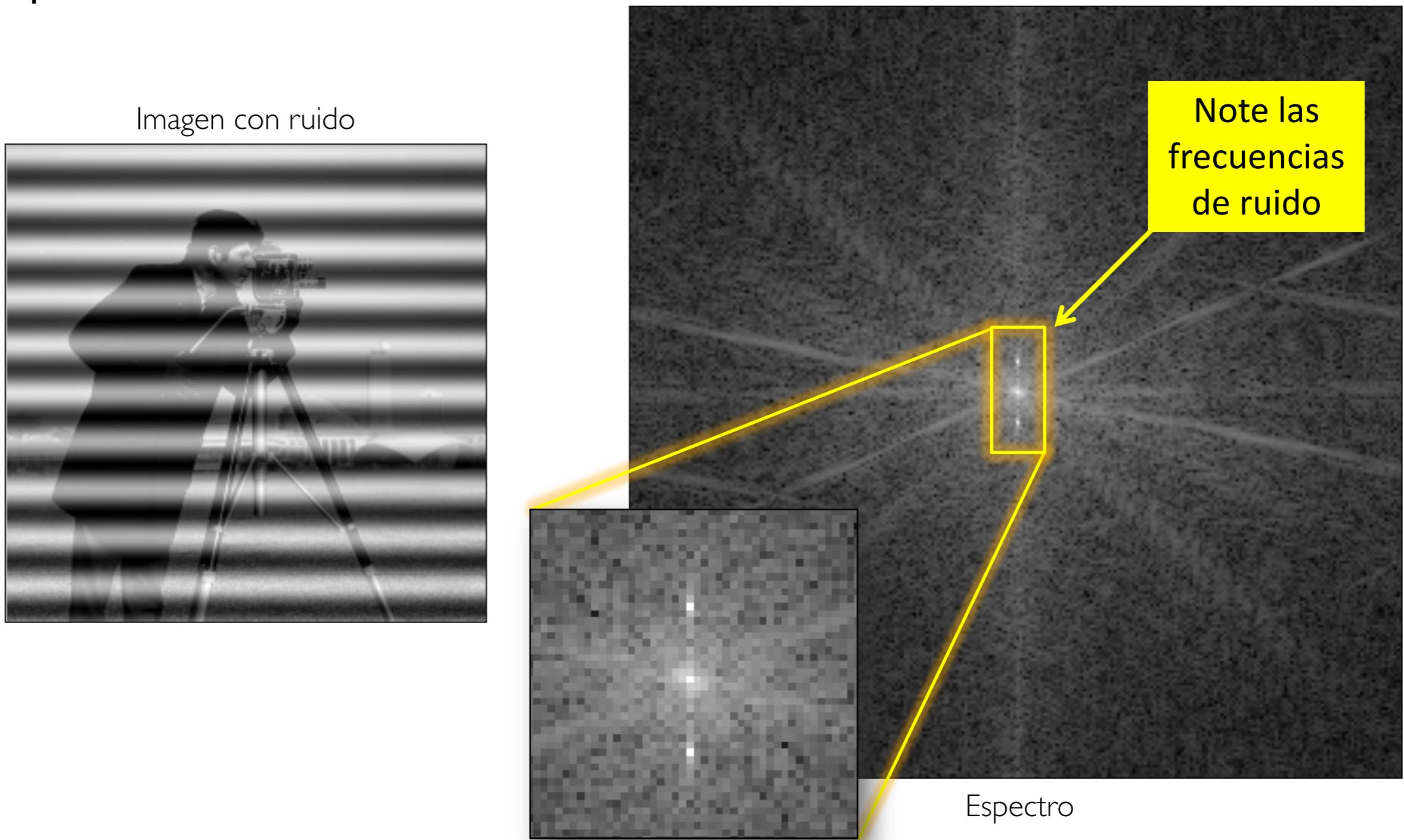


```
F = np.fft.fft2(noise_img)
fshift = np.fft.fftshift(F)
spectrum = 0.1*np.log(np.abs(fshift))

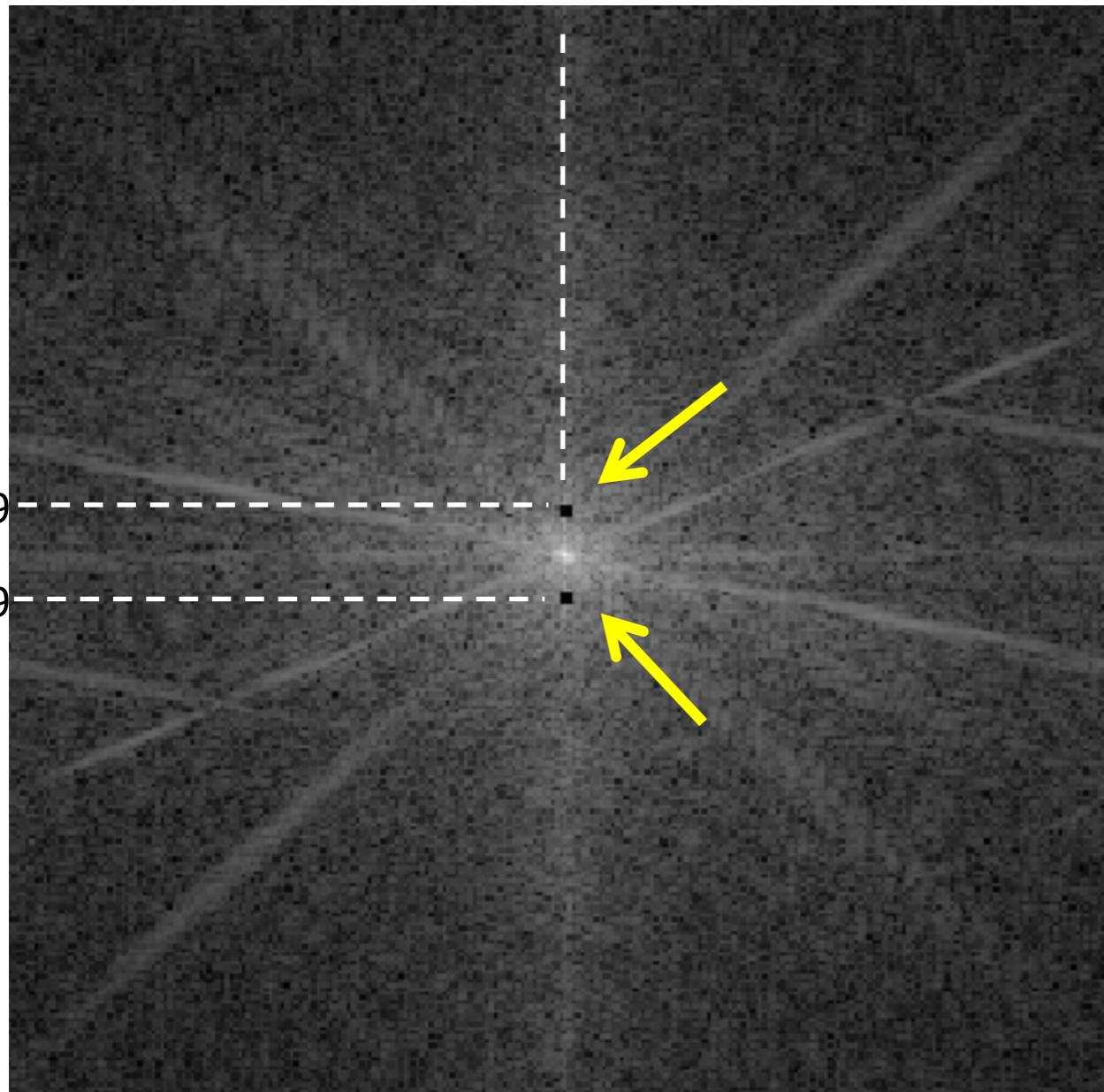
cv2.imshow('fft2', spectrum)
```

Usualmente se muestra el espectro con un logaritmo más 1 (POR QUÉ?)

- ▶ Calculemos el espectro de Fourier de la imagen con ruido, ¿qué notamos?



- ▶ Qué pasaría si borramos esas frecuencias



```
fshift[118:120,128:130]=0.0  
fshift[138:140,128:130]=0.0  
  
S= np.fft.ifft2(np.fft.fftshift(fshift))  
S= S.real  
  
cv2.imshow('output', S)  
cv2.waitKey(0)
```

- ▶ Wooow increíble



Imagen con ruido

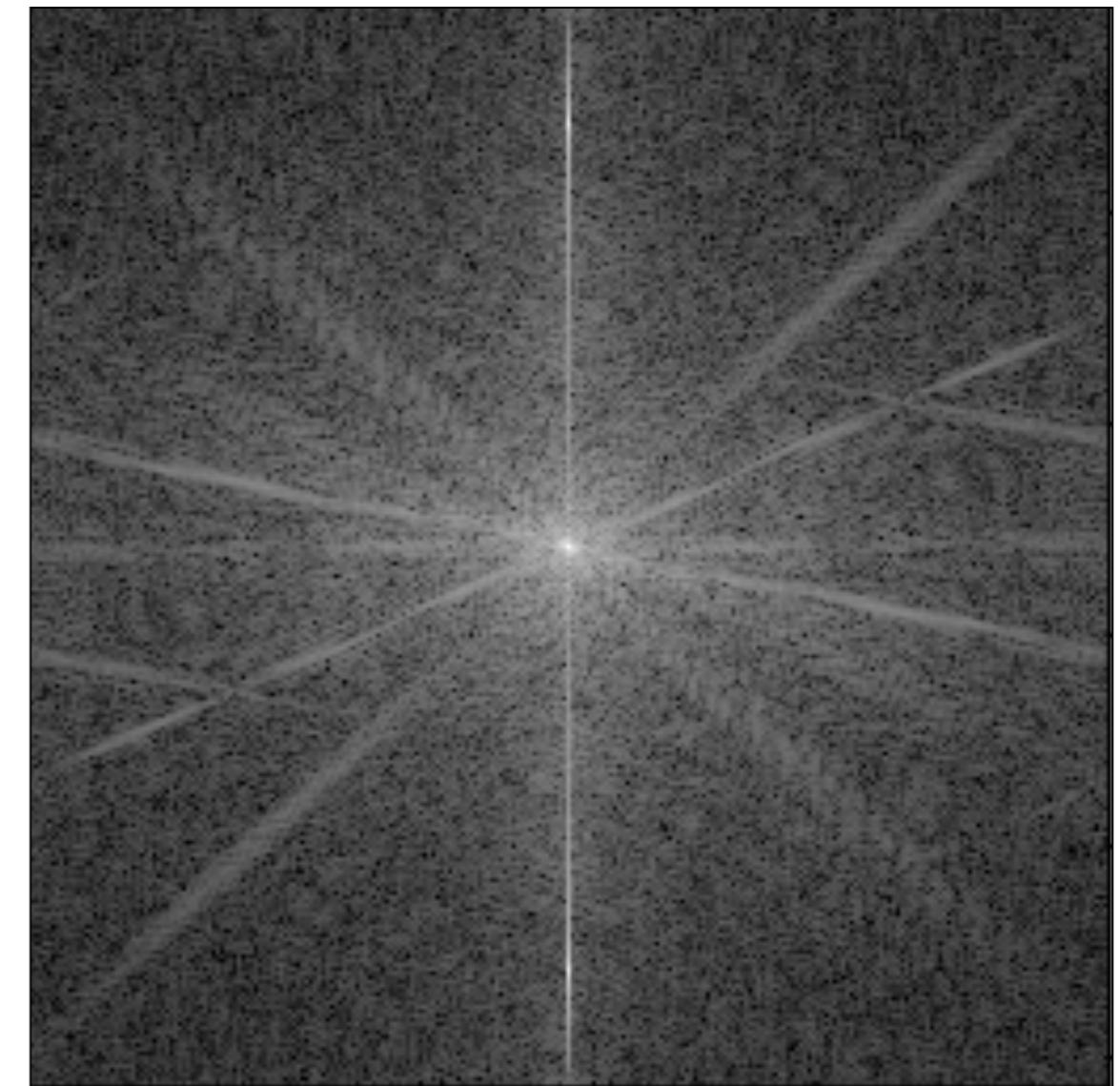


Imagen filtrada en el
espectro con Fourier

- ▶ Suponga que ahora hay más ruido, qué frecuencias elimino?

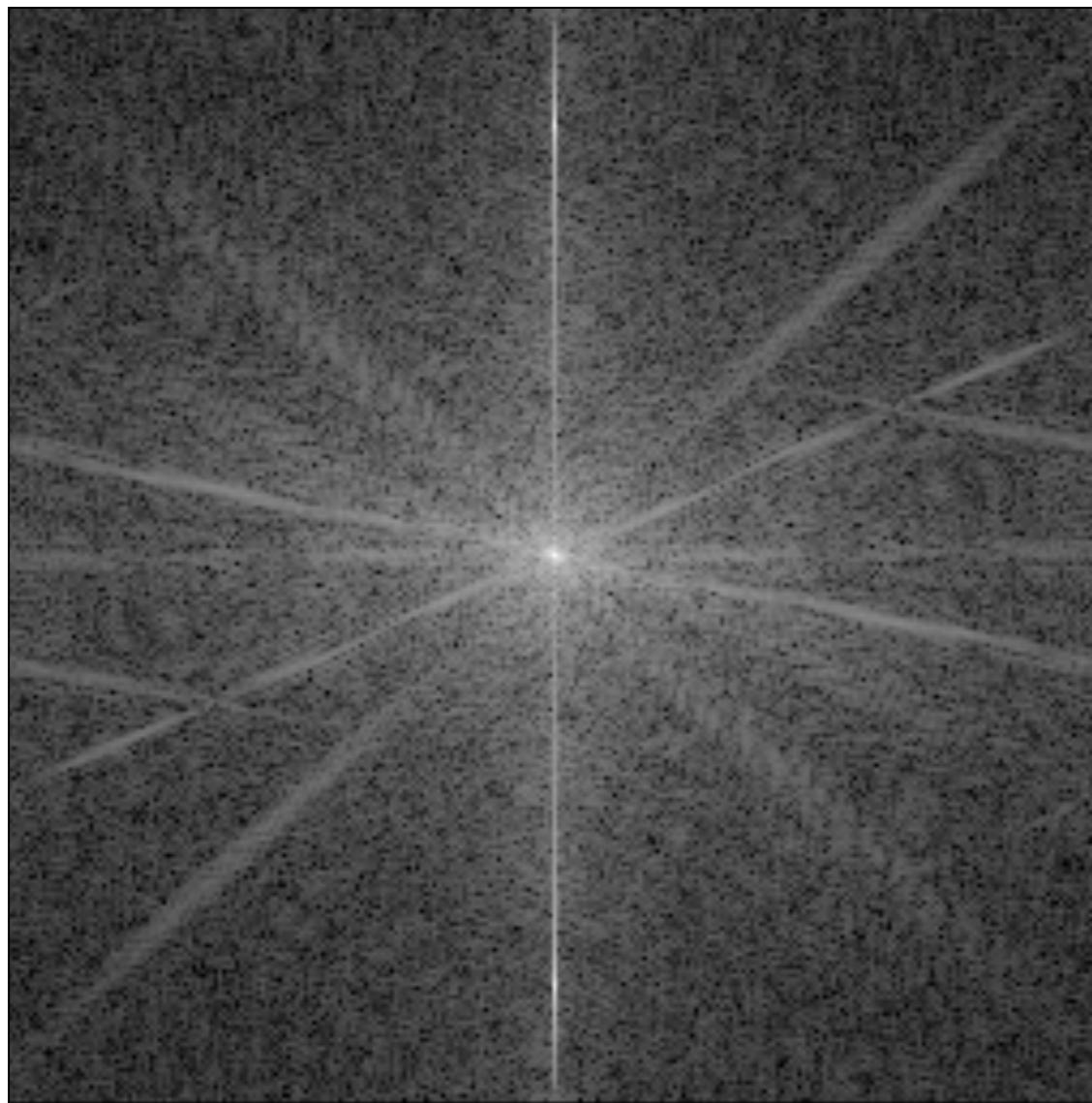


Imagen con ruido

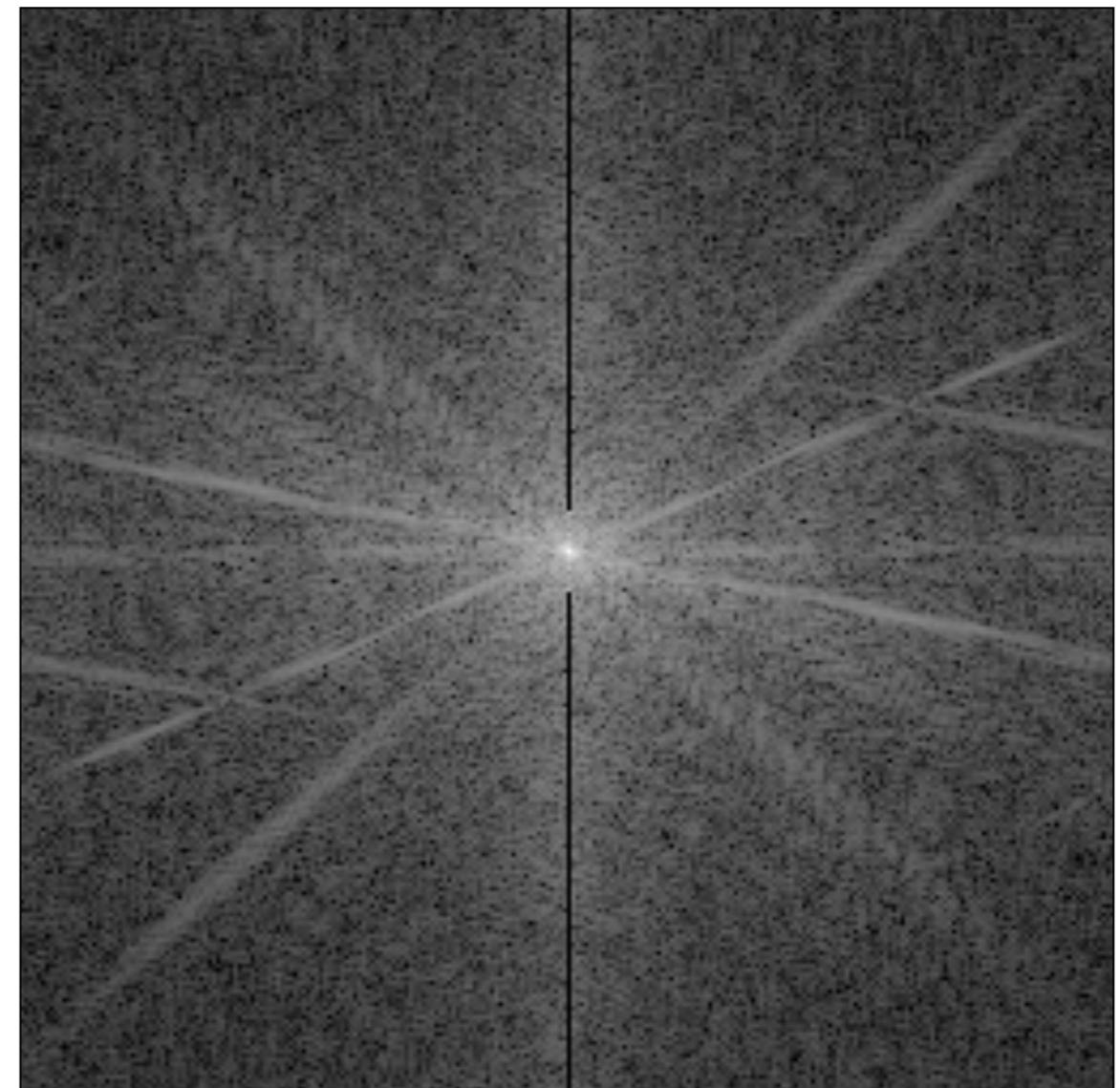


Espectro

- ▶ Si eliminamos al frecuencia que genera el ruido, qué sucede en el espacio



Espectro Original



Espectro Filtrado

- ▶ Resultado de eliminación del ruido

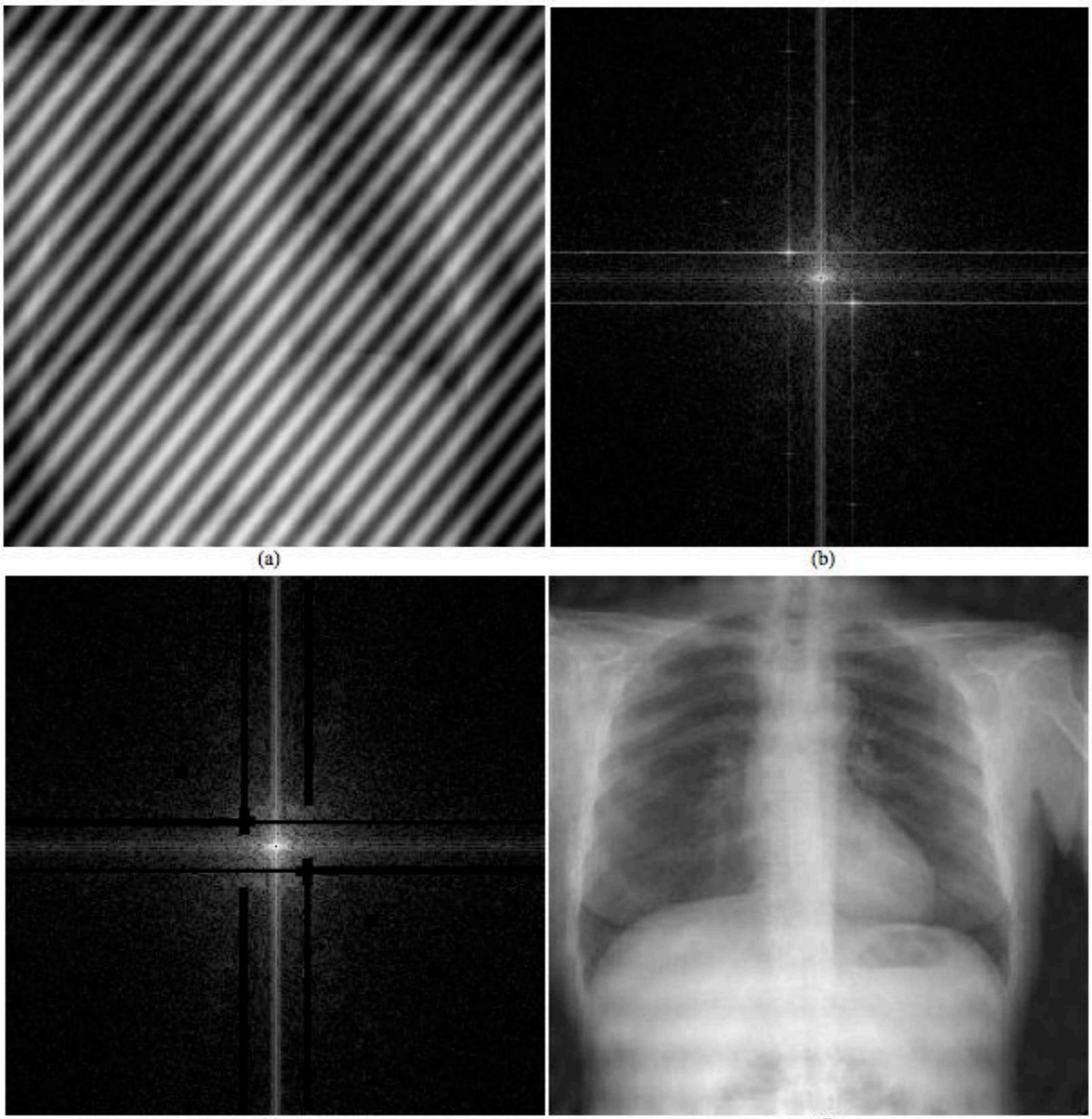


Imagen con ruido



Imagen filtrada en el
espectro con Fourier

Aplicacion 3

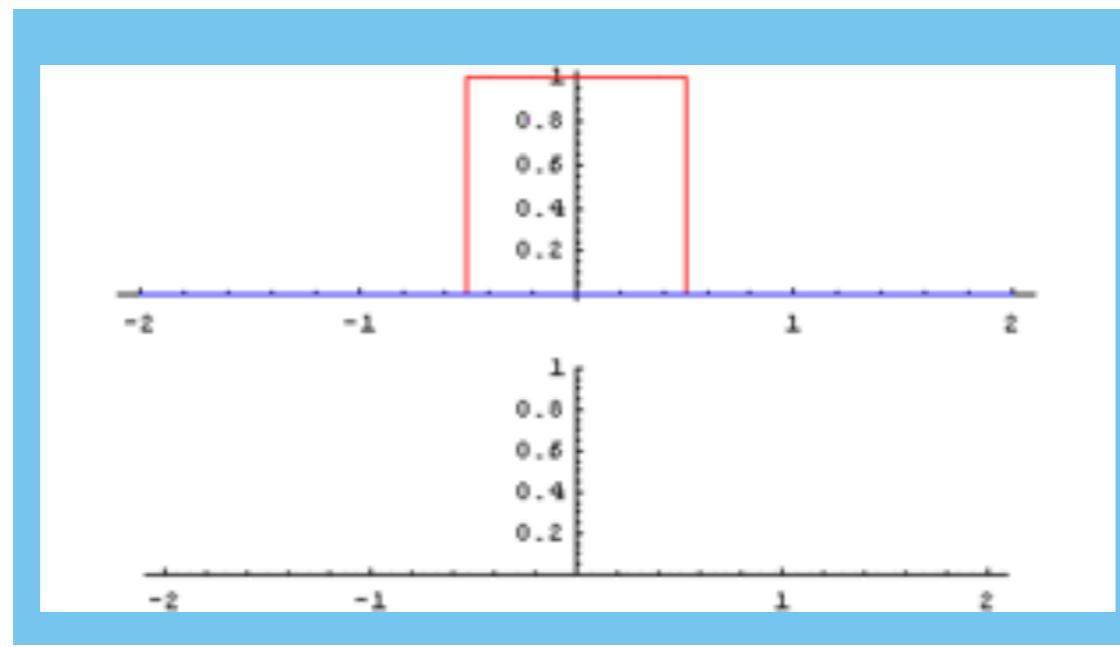


- Mejoramiento en la frecuencia
 - Convolución y Filtrado
 - Filtro Ideal
 - Filtro Gaussiano



► 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 la convolución

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

Transformada
de Fourier

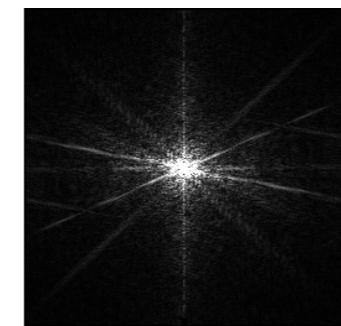
Multiplicación



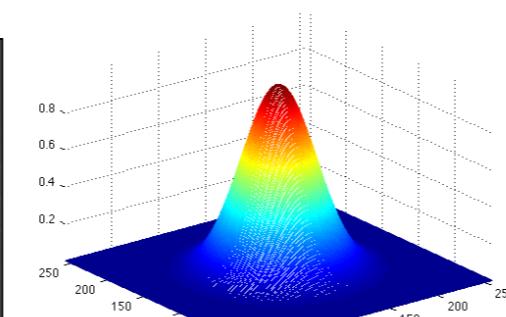
Proceso general



Imagen
Original
 $f_{i,j}$



Transformada
de Fourier



Filtrado con
función H



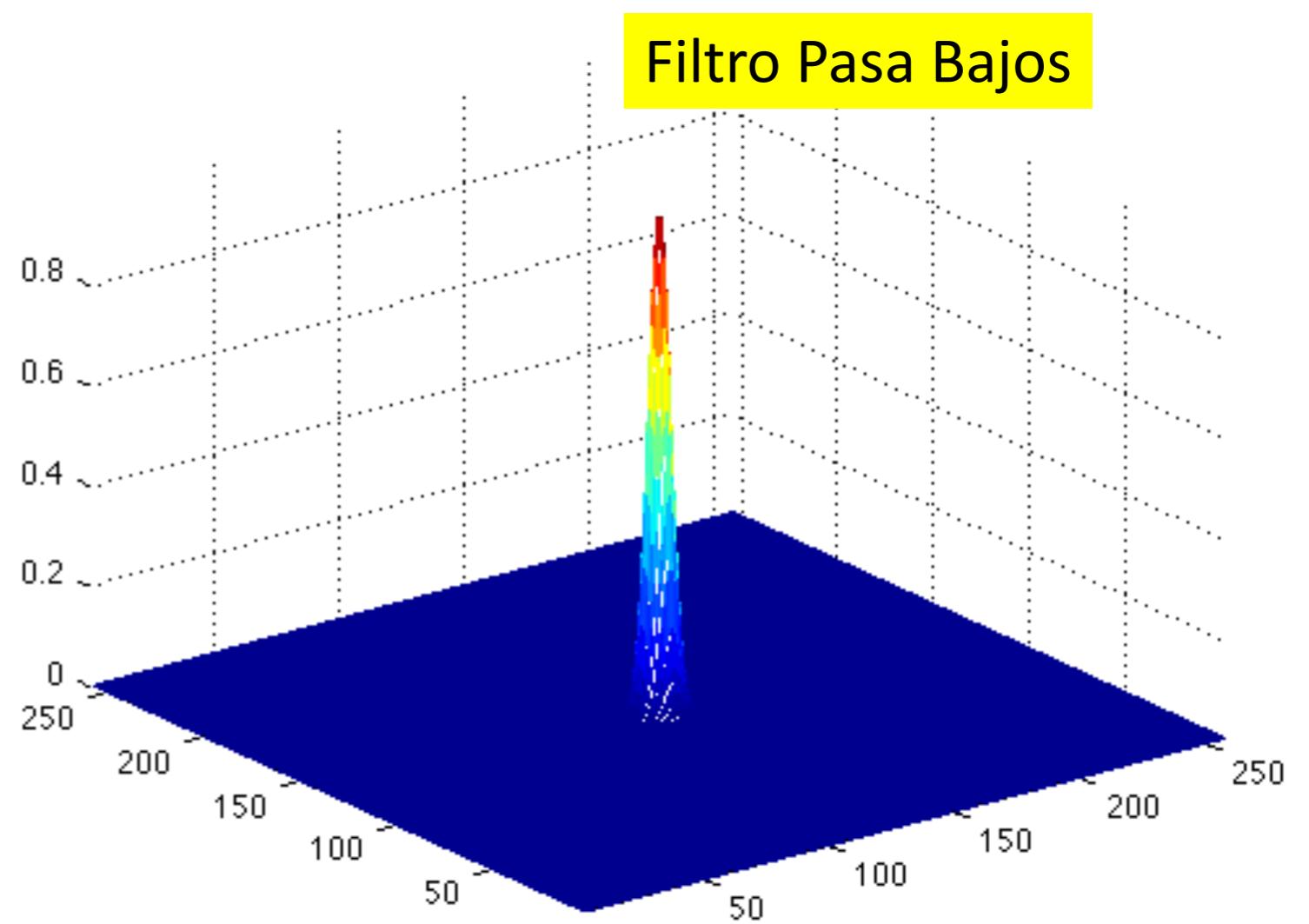
Imagen
Transformada
 $g_{i,j}$

$$H \cdot F$$

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

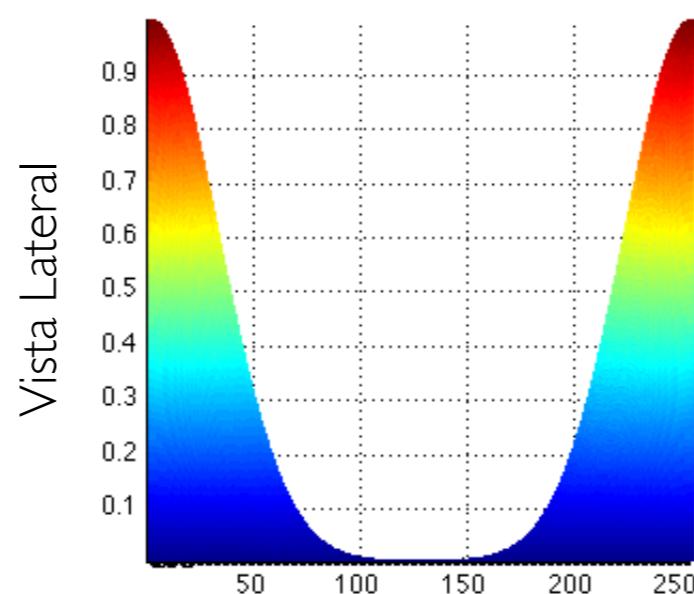
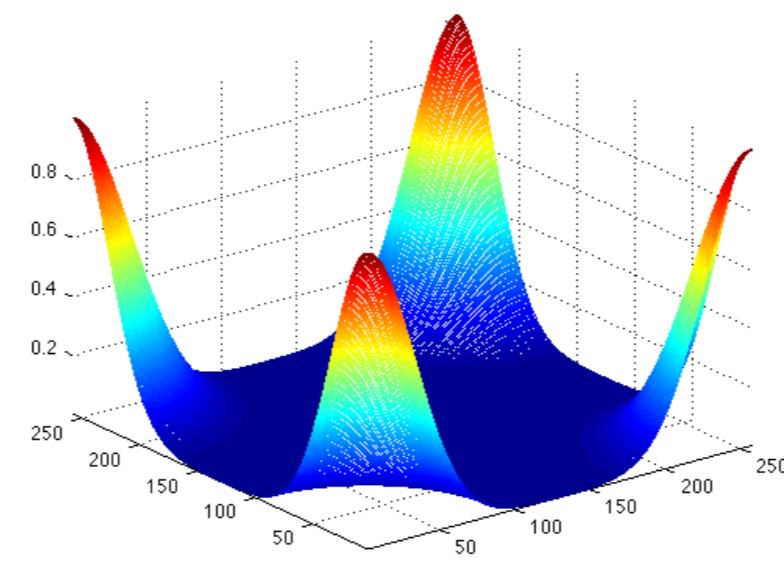
$$F(f)$$

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

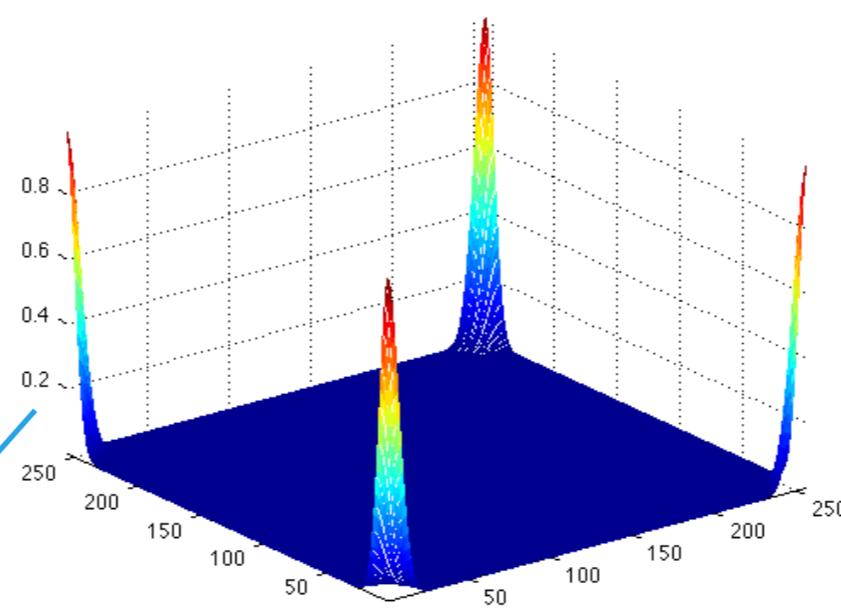
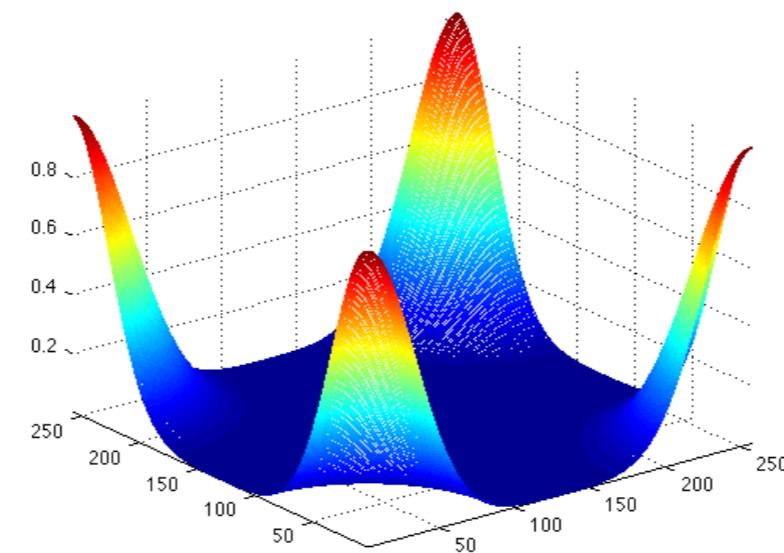


$$H(x, y) = e^{0.005 \cdot (x^2 + y^2)}$$

- ▶ Veamos distintos filtros Pasabajos



- ▶ Veamos distintos filtros Pasabajos



¿Hey, por qué se ve peor?