

In [301...]

```
import warnings
import numpy as np

warnings.filterwarnings('ignore')
```

4. Filtrado en el dominio de la frecuencia

4.1. Bases

4.1.1. Un poco de historia

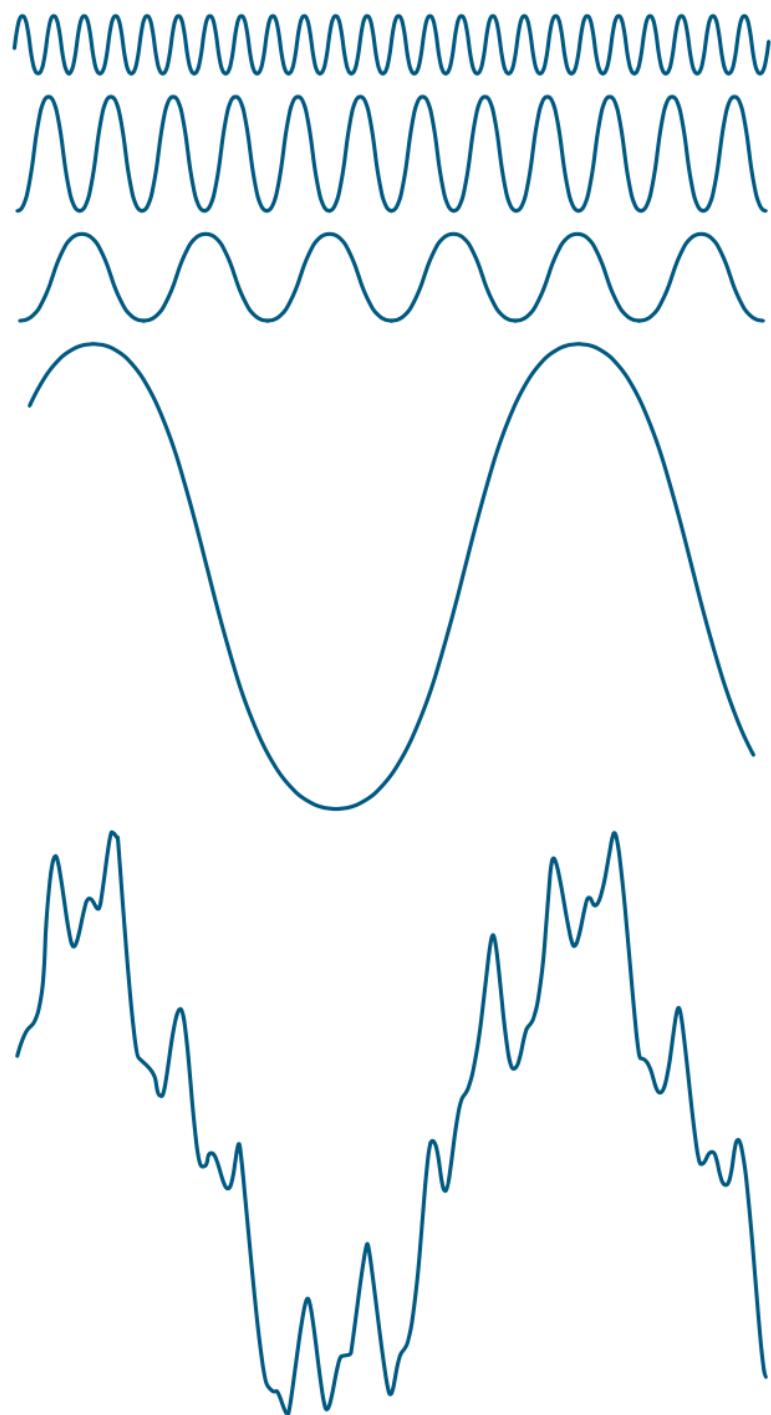
El matemático francés Jean Baptiste Joseph Fourier nació en 1768 en la ciudad de Auxerre, Francia.

La contribución por la que se le recuerda más fue esbozada en unas memorias de 1807 y posteriormente publicada en 1822 en su libro *La Théorie Analitique de la Chaleur* (*La teoría analítica del calor*).

- Este libro fue traducido al inglés 55 años después por Freeman.

Básicamente, la contribución de Fourier en este campo establece que cualquier función periódica se puede expresar como la suma de senos y/o cosenos de diferentes frecuencias, cada uno de ellos multiplicado por un coeficiente diferente (ahora llamamos a esta suma serie de Fourier).

- No importa cuán complicada sea la función; si es periódico y satisface algunas condiciones matemáticas suaves, puede representarse mediante dicha suma.
- Esto se da por sentado ahora, pero, en el momento en que apareció por primera vez, el concepto de que funciones complicadas podían representarse como una suma de senos y cosenos simples no era en absoluto intuitivo.



Las funciones que no son periódicas (pero cuyo área bajo la curva es finita) se pueden expresar como la integral de senos y/o cosenos multiplicada por una función de ponderación.

- La formulación en este caso es la transformada de Fourier, y su utilidad es incluso mayor que la serie de Fourier en muchas disciplinas teóricas y aplicadas.

Ambas representaciones comparten la importante característica de que una función, expresada en una serie de Fourier o en una transformada, puede reconstruirse (recuperarse) completamente mediante un proceso inverso, sin pérdida de información.

Esta es una de las características más importantes de estas representaciones porque nos permite trabajar en el dominio de Fourier (generalmente llamado dominio de la frecuencia) y luego regresar al dominio original de la función sin perder ninguna información.

La llegada de las computadoras digitales y el "descubrimiento" de un algoritmo de transformada rápida de Fourier (FFT) a principios de la década de 1960 revolucionaron el campo del procesamiento de señales.

- Estas dos tecnologías centrales permitieron por primera vez el procesamiento práctico de una serie de señales de excepcional importancia, desde monitores y escáneres médicos hasta comunicaciones electrónicas modernas.

4.2. Conceptos preliminares

4.2.1. Números complejos

Un número complejo C , está definido como $C = R + jI$, donde R e I son números reales, y $j = \sqrt{-1}$. Aquí, R denota la parte real del número complejo e I su parte imaginaria.

Los números reales son un subconjunto de números complejos en los que $I = 0$.

El conjugado de un número complejo C , denotado C^* , se define como $C^* = R - jI$

Los números complejos pueden verse geométricamente como puntos en un plano (llamado plano complejo) cuya abscisa es el eje real (valores de R) y cuya ordenada es el eje imaginario (valores de I).

Es decir, el número complejo $R + jI$ es el punto (R, I) en el sistema de coordenadas del plano complejo.

A veces es útil representar números complejos en coordenadas polares:

$C = |C|(\cos \theta + j \sin \theta)$, donde $|C| = \sqrt{R^2 + I^2}$ es la longitud del vector que se extiende desde el origen del plano complejo hasta el punto (R, I) , y θ es el ángulo entre el vector y el eje real.

Empleando la fórmula de Euler:

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

nos da la representación en coordenadas polares: $C = |C|e^{j\theta}$.

Las ecuaciones anteriores son aplicables también a funciones complejas.

Una función compleja, $F(u)$, de una variable real u , se puede expresar como la suma $F(u) = R(u) + jI(u)$, donde $R(u)$ e $I(u)$ son los valores reales y funciones componentes imaginarias de $F(u)$.

4.2.2. Series de Fourier

Como se indicó en la sección anterior, una función $f(t)$ de una variable continua, t , que es periódica con un período, T , se puede expresar como la suma de senos y cosenos multiplicada por coeficientes apropiados. Esta suma, conocida como serie de Fourier, tiene la forma

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{j t \frac{2\pi n}{T}}$$

donde

$$c_n = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-j t \frac{2\pi n}{T}} dt$$

son coeficientes.

4.2.3. Impulsos

4.2.4. La transformada de Fourier de funciones de una variable continua

La transformada de Fourier de una función continua $f(t)$ de una variable continua t , denotada como $\mathcal{I}\{f(t)\}$, está definida por la ecuación:

$$\mathcal{I}\{f(t)\} = F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j 2\pi \mu t} dt$$

donde μ es una variable continua.

Ya que t desaparece con la integración $\mathcal{I}\{f(t)\}$ es una función solo de μ . Esto significa que $\mathcal{I}\{f(t)\} = F(\mu)$.

Por el contrario, dada $F(\mu)$, podemos obtener $f(t)$ usando la *transformada inversa de Fourier*, descrita como:

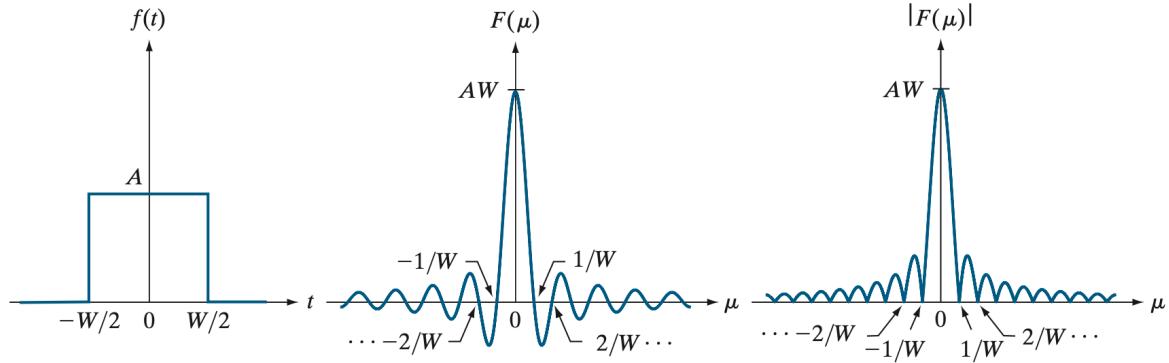
$$f(t) = \int_{-\infty}^{\infty} F(\mu) e^{j 2\pi \mu t} d\mu$$

donde hicimos uso del hecho de que la variable μ está integrada en la transformada inversa y escribimos simplemente $f(t)$.

Las ecuaciones anteriores comprenden el llamado par de transformadas de Fourier, a menudo denotado como $f(t) \Leftrightarrow F(\mu)$.

- La doble flecha indica que la expresión de la derecha se obtiene tomando la transformada de Fourier directa de la expresión de la izquierda, mientras que la expresión de la izquierda se obtiene tomando la transformada inversa de Fourier de la expresión de la derecha.

Ejemplo: La transformada de Fourier de la figura



$$\begin{aligned}
 F(\mu) &= \int_{-\infty}^{\infty} f(t)e^{-j2\pi\mu t}dt = \int_{-W/2}^{W/2} Ae^{-j2\pi\mu t}dt \\
 &= \frac{-A}{j2\pi\mu} [e^{-j2\pi\mu t}]_{-W/2}^{W/2} = \frac{-A}{j2\pi\mu} [e^{-j\pi\mu W} - e^{j\pi\mu W}] \\
 &= \frac{A}{j2\pi\mu} [e^{j\pi\mu W} - e^{-j\pi\mu W}] \\
 &= AW \frac{\sin(\pi\mu W)}{(\pi\mu W)}
 \end{aligned}$$

4.2.5. Convolución

Estamos interesados en la convolución de dos funciones continuas, $f(t)$ y $h(t)$, de una variable continua t . La convolución de estas dos funciones, denotada como antes por el operador \star , se define como:

$$(f \star h)(t) \equiv \int_{-\infty}^{\infty} f(\tau)h(t - \tau)d\tau$$

de aquí se sigue que, si definimos $H(\mu) = \mathcal{I}\{h(t)\}$, entonces:

$$\mathcal{I}\{(h \star f)(t)\} = (H \cdot F)(\mu)$$

donde " \cdot " indica multiplicación.

Si nos referimos al dominio de t como dominio espacial y al dominio de μ como dominio de frecuencia, la ecuación anterior nos dice que la transformada de Fourier de la convolución de dos funciones en el dominio espacial es igual a la producto en el dominio de la frecuencia de las transformadas de Fourier de las dos funciones.

Por el contrario, si tenemos el producto de las dos transformadas, podemos obtener la convolución en el dominio espacial calculando la transformada inversa de Fourier.

Es decir, $f \star h$ y $H \cdot F$ son un par de transformadas de Fourier.

$$(f \star h)(t) \Leftrightarrow (H \cdot F)(\mu)$$

Este resultado es la mitad del teorema de convolución.

Siguiendo un desarrollo similar resultaría en la otra mitad del teorema de convolución:

$$(f \cdot h)(t) \Leftrightarrow (H \star F)(\mu)$$

que establece que la convolución en el dominio de la frecuencia es análoga a la multiplicación en el dominio espacial, estando ambas relacionadas por las transformadas de Fourier directa e inversa, respectivamente.

4.3. Muestreo y la transformada de Fourier de funciones muestreadas

En esta parte utilizamos los conceptos de la sección anterior para formular una base para expresar matemáticamente el muestreo. Partiendo de principios básicos, esto nos llevará a la transformada discreta de Fourier.

4.3.1. Muestreo

Las funciones continuas deben convertirse en una secuencia de valores discretos antes de poder procesarlas en una computadora.

- Esto requiere muestreo y cuantificación.
- Considere una función continua, $f(t)$, que deseamos muestrear a intervalos uniformes, ΔT , de la variable independiente t (ver figura).
- Inicialmente suponemos que la función se extiende desde $-\infty$ hasta $+\infty$ con respecto a t .

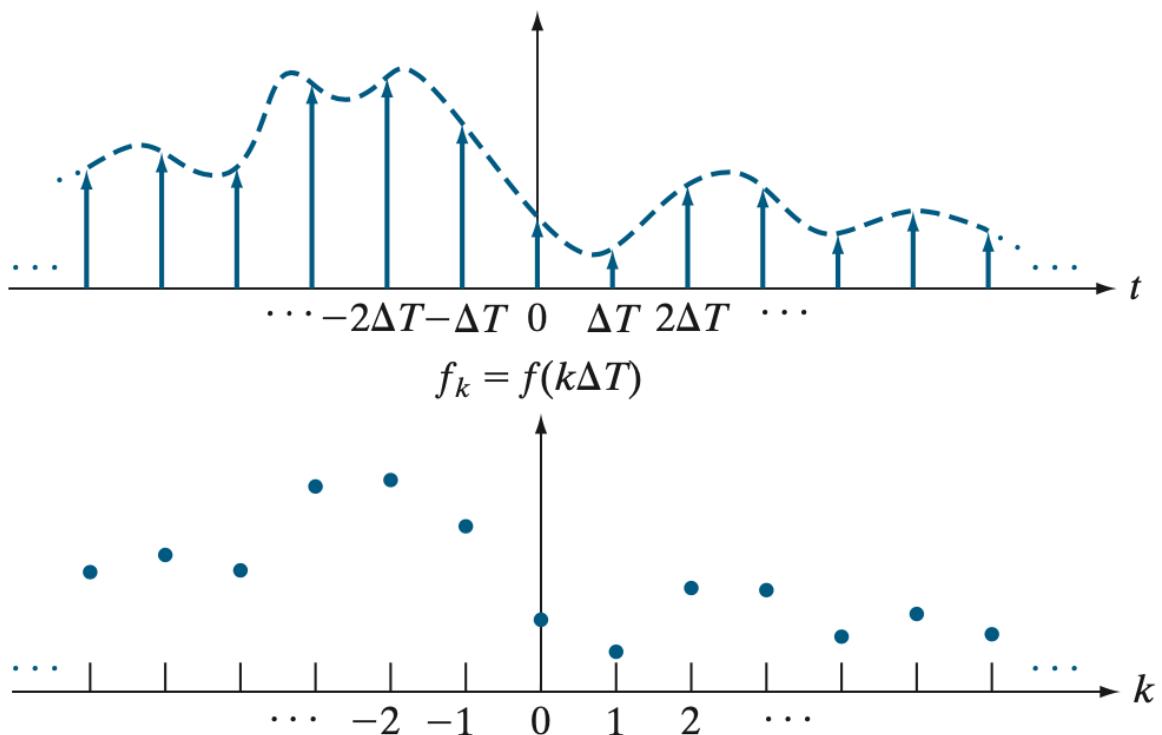
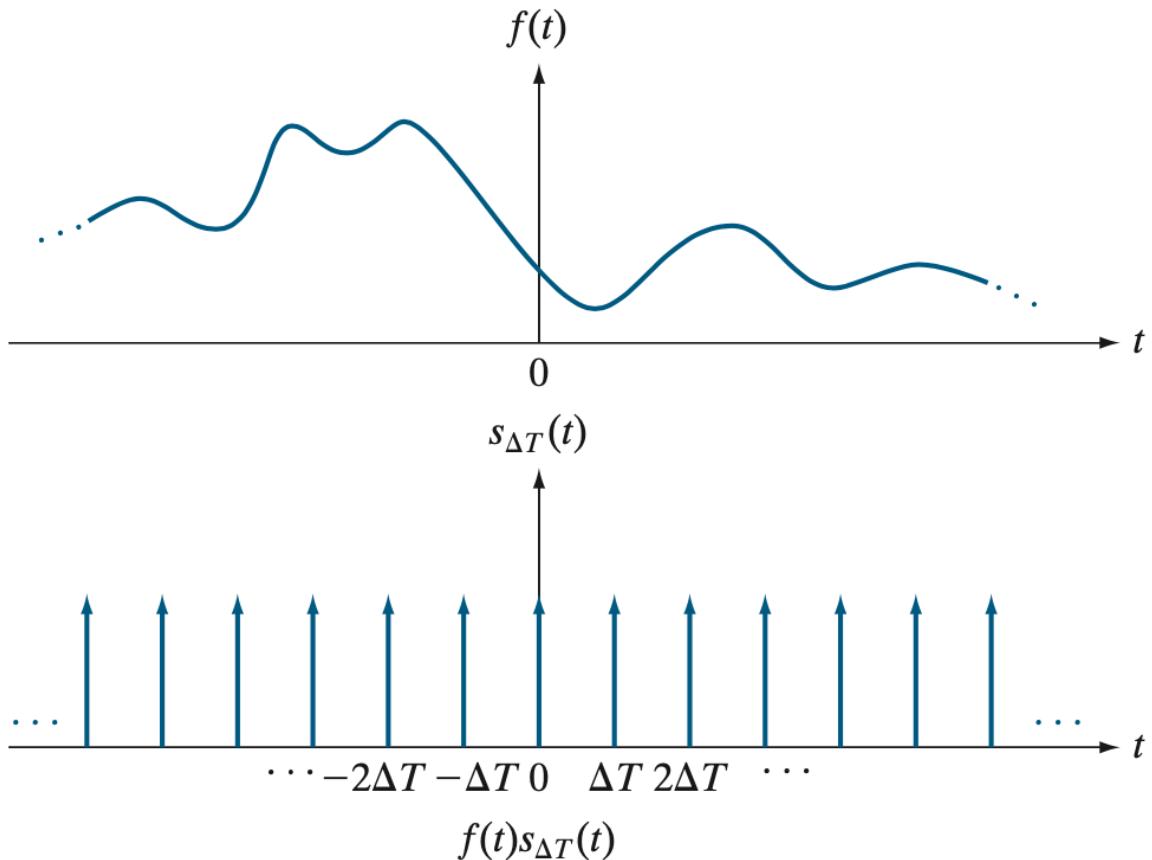
Una forma de modelar el muestreo es multiplicar $f(t)$ por una función de muestreo igual a un tren de impulsos separados por ΔT unidades:

$$\tilde{f}(t) = f(t)s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)$$

donde $\tilde{f}(t)$ denota la función muestreada.

- Cada componente de esta suma es un impulso ponderado por el valor de $f(t)$ en la ubicación del impulso, como muestra la figura.

- El valor de cada muestra viene dado por la "fuerza" del impulso ponderado.



4.3.2. Transformada de Fourier de funciones muestreadas

Sea $F(\mu)$ la transformada de Fourier de una función continua $f(t)$.

- Como se analizó anteriormente, la función muestreada correspondiente, $\tilde{f}(t)$, es el producto de $f(t)$ y un tren de impulsos.
- Sabemos por el teorema de convolución que la transformada de Fourier del producto de dos funciones en el dominio del espacio es la convolución de las transformadas de las dos funciones en el dominio de la frecuencia.

Así, la transformada de Fourier de la función muestreada es:

$$\begin{aligned}\tilde{F}(\mu) &= \mathcal{I}\{\tilde{f}(t)\} = \mathcal{I}\{f(t)s_{\Delta T}(t)\} \\ &= (F \star S)(\mu)\end{aligned}$$

dónde (del ejemplo anterior):

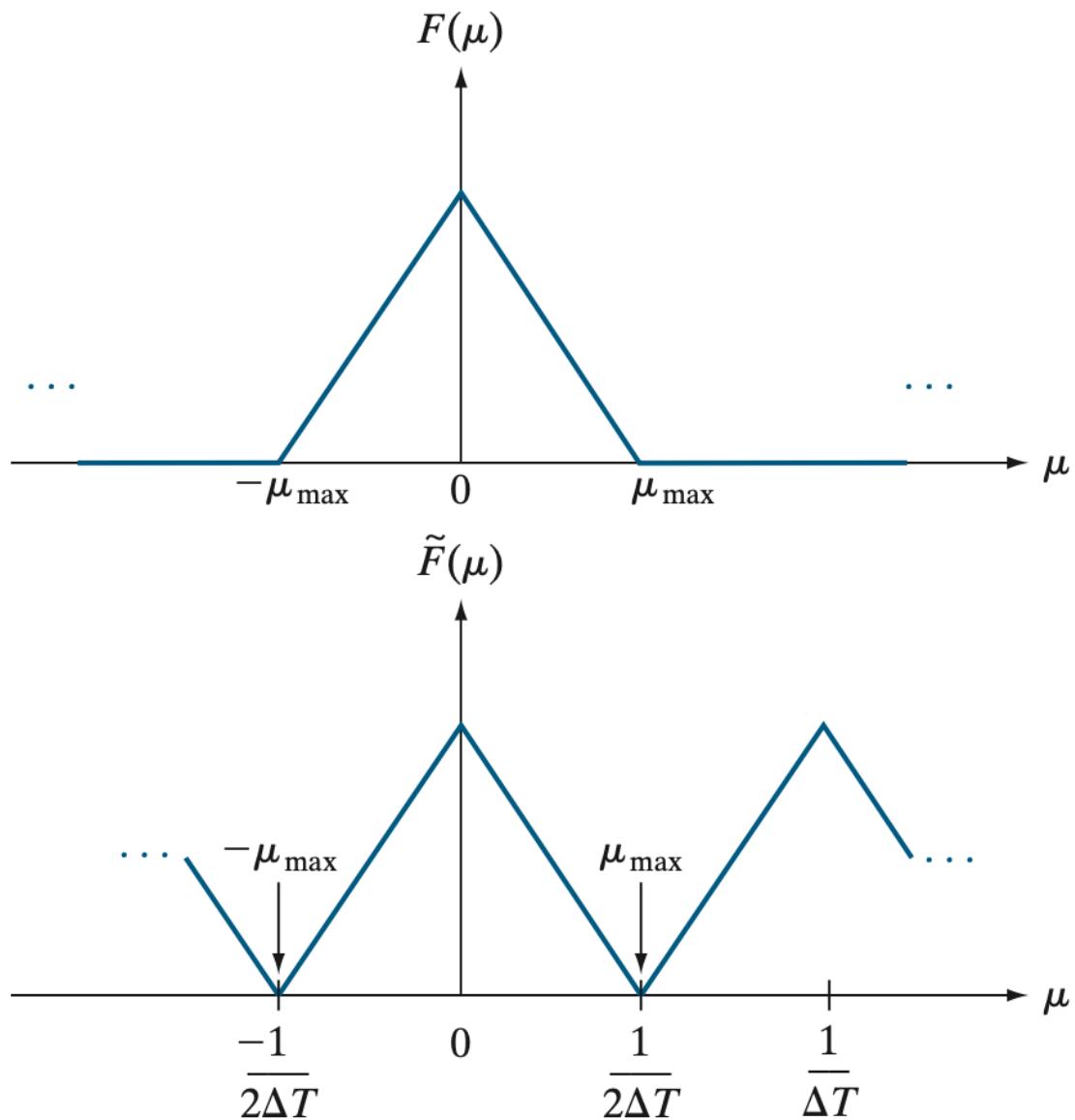
$$S(\mu) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \delta\left(\mu - \frac{n}{\Delta T}\right)$$

es la transformada de Fourier el tren de impulsos $s_{\Delta T}(t)$.

4.3.3. El teorema del muestreo

Una función $f(t)$ cuya transformada de Fourier es cero para valores de frecuencias fuera de un intervalo finito (banda) $[-\mu_{max}, \mu_{max}]$ alrededor del origen se llama función de banda limitada.

Como se muestra en la figura, un valor más alto de ΔT haría que los períodos en $F(\mu)$ se fusionaran; un valor más bajo proporcionaría una separación clara entre los períodos.



Podemos recuperar $f(t)$ a partir de sus muestras si podemos aislar una única copia de $F(\mu)$ de la secuencia periódica de copias de esta función contenida en $\tilde{F}(\mu)$, la transformada de la función muestreada $\tilde{f}(t)$.

$\tilde{F}(\mu)$ es una función periódica continua con período $1/\Delta T$. Por lo tanto, todo lo que necesitamos es un período completo para caracterizar toda la transformación.

- En otras palabras, podemos recuperar $f(t)$ de ese único período tomando su transformada inversa de Fourier.
- Es posible extraer de $\tilde{F}(\mu)$ un único período que sea igual a $F(\mu)$ si la separación entre copias es suficiente, es decir si

$$\frac{1}{\Delta T} > 2\mu_{\max}$$

Esta ecuación indica que una función continua de banda limitada se puede recuperar completamente a partir de un conjunto de sus muestras si las muestras se adquieren a una velocidad que excede el doble del contenido de frecuencia más alto de la función.

- Esto se conoce como *el teorema de muestreo*.

4.4. La transformada discreta de Fourier (DFT) de una variable

La transformada de Fourier de una función muestreada de banda limitada que se extiende desde $-\infty$ a ∞ es una función continua y periódica que también se extiende desde $-\infty$ a ∞ .

- En la práctica, trabajamos con un número finito de muestras por lo que debemos encontrar la DFT de dichos conjuntos de muestras finitos.

Tenemos una expresión para la transformada $\tilde{F}(\mu)$, de los datos muestreados en términos de la transformación de la función original.

- Encontramos esa expresión a partir de la definición de la transformada de Fourier:

$$\begin{aligned}\tilde{F}(\mu) &= \int_{-\infty}^{\infty} \tilde{f}(t) e^{-j2\pi\mu t} dt = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(t) \delta(t - n\Delta T) e^{-j2\pi\mu t} dt \\ &= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) \delta(t - n\Delta T) e^{-j2\pi\mu t} dt \\ &= \sum_{n=-\infty}^{\infty} f_n e^{-j2\pi\mu n\Delta T}\end{aligned}$$

Como solamente tenemos un número finito de muestras (M), entonces supongamos que queremos obtener M muestras equiespaciadas de $F(\mu)$ tomadas durante el intervalo de un período desde $\mu = 0$ hasta $\mu = 1/\Delta T$. Esto se logra mediante tomar las muestras a las siguientes frecuencias:

$$\mu = \frac{m}{M\Delta T} \quad m = 0, 1, 2, \dots, M-1$$

Sustituyendo en la ecuación previa:

$$F_m = \sum_{n=0}^{M-1} f_n e^{-j2\pi m n / M} \quad m = 0, 1, 2, \dots, M-1$$

Esta expresión es la transformada discreta de Fourier que estamos buscando.

- Dado un conjunto $\{f_m\}$ que consta de M muestras de $f(t)$, la ecuación produce un conjunto $\{F_m\}$ de M valores complejos correspondientes a la transformada discreta de Fourier del conjunto de muestras de entrada.

Por el contrario, dado $\{F_m\}$, podemos recuperar el conjunto de muestra $\{f_m\}$ usando la transformada de Fourier discreta inversa (IDFT):

$$f_n = \frac{1}{M} \sum_{m=0}^{M-1} F_m e^{j2\pi mn/M} \quad n = 0, 1, 2, \dots, M-1$$

Usamos m y n en el desarrollo anterior para denotar variables discretas porque es típico hacerlo para derivaciones. Sin embargo, es más intuitivo, especialmente en dos dimensiones, usar la notación x e y para variables de coordenadas de imagen y u y v para variables de frecuencia, donde se entiende que son números enteros.

Entonces, las ecuaciones anteriores se convierten en:

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M} \quad u = 0, 1, 2, \dots, M-1$$

y

$$f(x) = \frac{1}{M} \sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M} \quad x = 0, 1, 2, \dots, M-1$$

Se puede demostrar que tanto la transformada discreta directa como la inversa son infinitamente periódicas, con período M . Es decir:

$$F(u) = F(u + kM)$$

y

$$f(x) = f(x + kM)$$

donde k es un número entero.

El equivalente discreto de la convolución 1-D es:

$$f(x) * h(x) = \sum_{m=0}^{M-1} f(m) h(x-m) \quad x = 0, 1, 2, \dots, M-1$$

4.5. Extensión a funciones de dos variables

4.5.1. El impulso 2D

El impulso, $\delta(t, z)$, de dos variables continuas, t y z , se define como antes:

$$\delta(t, z) = \begin{cases} 1 & \text{if } t = z = 0 \\ 0 & \text{otherwise} \end{cases}$$

y

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(t, z) dt dz = 1$$

Como en el caso 1-D, el impulso 2-D exhibe la propiedad de corrimiento bajo integración:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t, z) \delta(t, z) dt dz = f(0, 0)$$

o más general, el impulso localizado en (t_0, z_0) :

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t, z) \delta(t - t_0, z - z_0) dt dz = f(t_0, z_0)$$

Para variables discretas x y y , el impulso 2-D se define como:

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y = 0 \\ 0 & \text{dof} \end{cases}$$

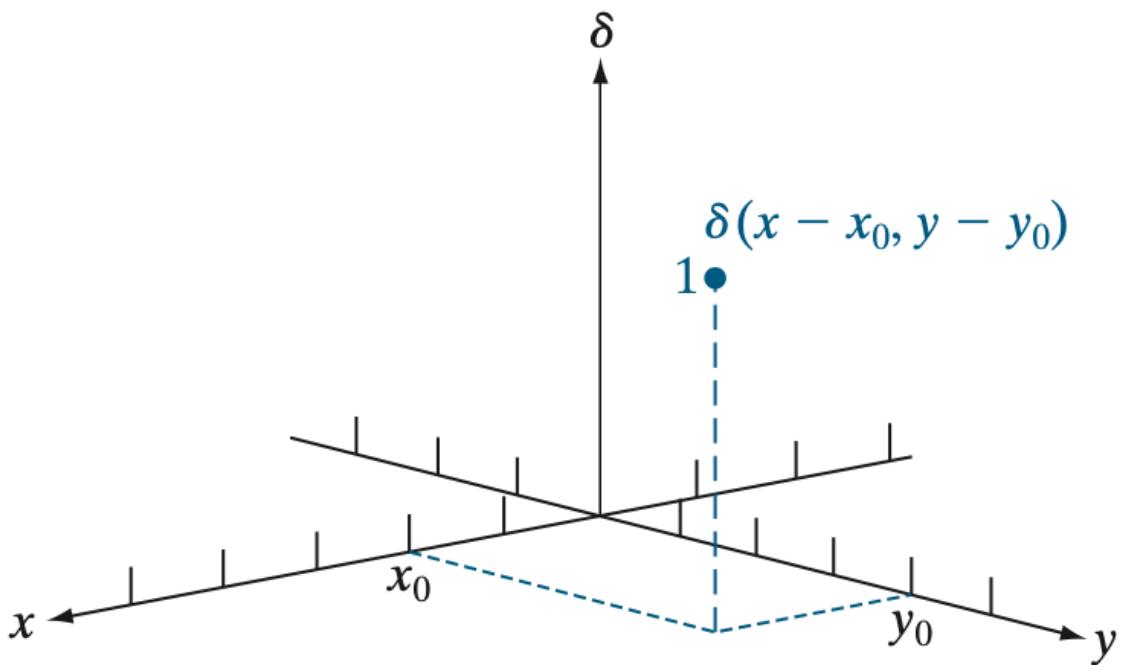
y la propiedad de corrimiento es:

$$\sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} f(x, y) \delta(x, y) = f(0, 0)$$

donde $f(x, y)$ es una función de las variables discretas x e y .

Para un impulso ubicado en las coordenadas (x_0, y_0) , la propiedad de corrimiento es:

$$\sum_{x=-\infty}^{\infty} \sum_{y=-\infty}^{\infty} f(x, y) \delta(x - x_0, y - y_0) = f(x_0, y_0)$$



4.5.2. La transformada de Fourier 2D continua

Sea $f(t, z)$ una función continua de dos variables continuas, t y z . El par de transformadas de Fourier continuas y bidimensionales viene dado por las expresiones

$$F(\mu, \nu) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t, z) e^{-j2\pi(\mu t + \nu z)} dt dz$$

y

$$f(t, z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\mu, \nu) e^{j2\pi(\mu t + \nu z)} d\mu d\nu$$

donde μ y ν son las variables de frecuencia. Cuando se hace referencia a imágenes, t y z se interpretan como variables espaciales continuas. Como en el caso 1-D, el dominio de las variables μ y ν define el dominio de frecuencia continua.

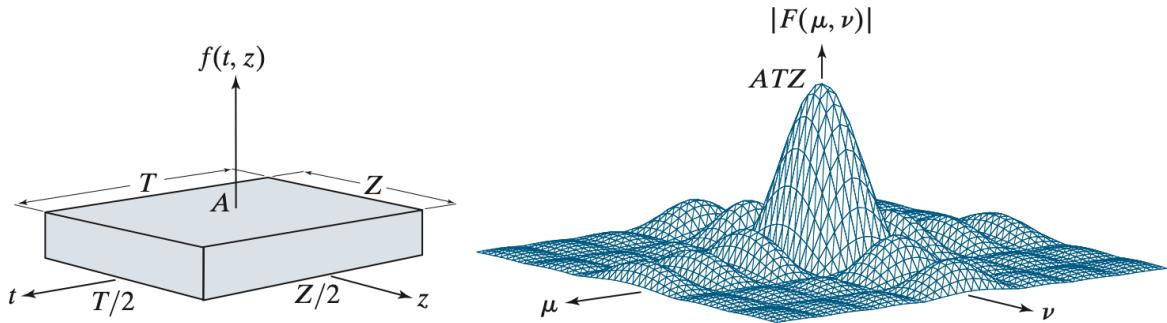
4.5.3. Muestreo 2D y el teorema de muestreo en 2D

De manera similar al caso 1-D, el muestreo en dos dimensiones se puede modelar usando una función de muestreo (es decir, un tren de impulsos 2-D):

$$s_{\Delta T \Delta Z}(t, z) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(t - m\Delta T, z - n\Delta Z)$$

donde ΔT y ΔZ son las separaciones entre muestras a lo largo de los ejes t y z de la función continua $f(t, z)$. La ecuación (4-61) describe un conjunto de impulsos periódicos

que se extienden infinitamente a lo largo de los dos ejes (ver figura). Como en el caso 1-D, multiplicar $f(t, z)$ por $s_{\Delta T \Delta Z}(t, z)$ produce la función muestreada.



4.5.5. La DFT en 2D y su inversa

Un desarrollo similar que en 1-D produciría la siguiente transformada de Fourier discreta (DFT) 2-D:

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

donde $f(x, y)$ es una imagen digital de tamaño $M \times N$. Como en el caso 1-D, la ecuación debe evaluarse para valores de las variables discretas u y v en los rangos $u = 0, 1, 2, \dots, M - 1$ y $v = 0, 1, 2, \dots, N - 1$.

Dada la transformada $F(u, v)$, podemos obtener $f(x, y)$ usando la transformada discreta inversa de Fourier (IDFT):

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

para $x = 0, 1, 2, \dots, M - 1$ y $y = 0, 1, 2, \dots, N - 1$.

Como en el caso 1-D, las ecuaciones anteriores constituyen un par de transformada de Fourier discreta 2-D, $f(x, y) \Leftrightarrow F(u, v)$.

In [302...]

```
import matplotlib.pyplot as plt

def show_images(*images, titles=[], cols=2, cmap='gray', figsize=(10, 10)):
    """
    Despliega una lista de imágenes en una cuadricula.
    Args:
        images (list): lista de imágenes.
        titles (list): lista de títulos para las imágenes (default=[]).
        cols (int): número de columnas de la cuadricula (default=2).
        cmap (str): mapa de color para las imágenes (default='gray').
        figsize (tuple): tamaño de la figura (default=(10, 10)).
    """
    # Implementation of show_images function
```

```

rows = len(images) // cols + (1 if len(images) % cols else 0)
fig, axes = plt.subplots(rows, cols, figsize=figsize)
for i, ax in enumerate(axes.flat):
    if i < len(images):
        ax.imshow(images[i], cmap=cmap)
        if titles is not None:
            ax.set_title(titles[i] if len(titles) > i else 'Figura {}'.format(i))
    ax.axis('off')
plt.tight_layout()
plt.show()

```

Implementación de la DFT e IDFT 2D de acuerdo a las ecuaciones anteriores. Tiene una complejidad computacional alta ($\mathcal{O}(n^4)$)

In [303...]

```

def dft2d(img):
    """
    Calcular la transformada de Fourier de una imagen.
    Args:
        img (np.ndarray): La imagen de entrada.
    Returns:
        np.ndarray: La transformada de Fourier de la imagen.
    """
    # obtener dimensiones de la imagen
    M, N = img.shape
    # inicializar la matriz de salida
    out = np.zeros((M, N), dtype=np.complex64)
    # calcular la dft-2d
    for u in range(M):
        for v in range(N):
            for x in range(M):
                for y in range(N):
                    out[u, v] += img[x, y] * np.exp(-1j * 2 * np.pi *
                        ((u * x) / M + (v * y) / N))
    # retornar la matriz de salida
    return out

```

In [304...]

```

def idft2d(img):
    """
    Calcular la transformada inversa de Fourier de una imagen.
    Args:
        img (np.ndarray): La imagen de entrada.
    Returns:
        np.ndarray: La transformada inversa de Fourier de la imagen.
    """
    # obtener dimensiones de la imagen
    M, N = img.shape
    # inicializar la matriz de salida
    out = np.zeros((M, N), dtype=np.complex64)
    # calcular la idft-2d
    for x in range(M):
        for y in range(N):
            for u in range(M):
                for v in range(N):
                    out[x, y] += img[u, v] * np.exp(1j * 2 * np.pi *
                        ((u * x) / M + (v * y) / N))
            out[x, y] /= M * N
    # retornar la matriz de salida
    return out

```

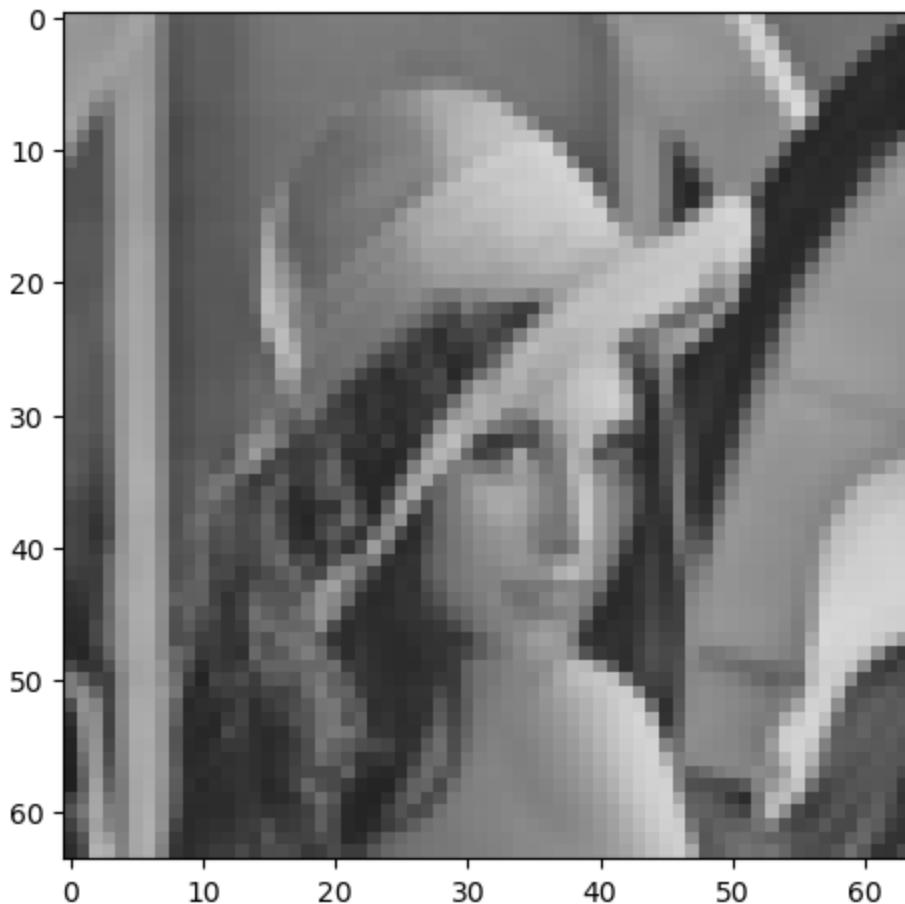
Vamos a probar con una imagen pequeña, de 64x64, para que no tarde demasiado.

In [305...]

```
from skimage.io import imread, imshow
from skimage.transform import resize
```

```
lena = imread('figs/lena.jpg', as_gray=True)
lena_small = resize(lena, (64, 64), anti_aliasing=True)
imshow(lena_small)
```

Out[305]: <matplotlib.image.AxesImage at 0x1411bc250>



Aplicamos la DFT implementada y con la que nos proporciona NumPy (Transformada Rápida de Fourier - que es tema de otro tipo de curso)

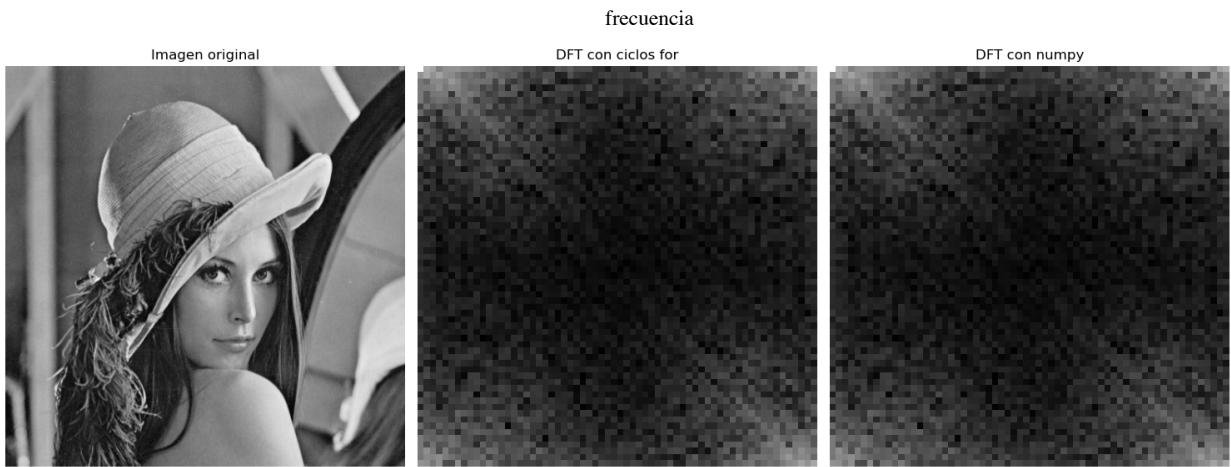
In [306...]

```
lena_dft = dft2d(lena_small)
lena_dft_np = np.fft.fft2(lena_small)
```

Vemos que ambas implementaciones dan el mismo resultado.

In [307...]

```
show_images(lena, np.log(1 + np.abs(lena_dft)), np.log(1 + np.abs(lena_dft_np))
           titles=['Imagen original', 'DFT con ciclos for', 'DFT con numpy'],
           cols=3, figsize=(15, 15))
```



Hacemos lo mismo con la transformada inversa.

```
In [308...]: lena_idft = idft2d(lena_dft)
lena_idft_np = np.fft.ifft2(lena_dft)
```

```
In [309...]: show_images(lena_small, np.abs(lena_idft), np.abs(lena_idft_np),
titles=['Imagen original', 'IDFT con ciclos for', 'IDFT con numpy'],
cols=3, figsize=(15, 15))
```



Ahora, solo con fines demostrativos calculamos la DFT de la imagen con su resolución original.

```
In [310...]: lena_dft = np.fft.fft2(lena)

show_images(lena, np.log(1 + np.abs(lena_dft)),
titles=['Imagen original', 'DFT con numpy'],
cols=2, figsize=(10, 10))
```



4.6. Propiedades de las DFT y IDFT en 2D

4.6.1. Espectro de Fourier y el ángulo de fase

Debido a que la DFT 2-D es compleja en general, se puede expresar en forma polar:

$$\begin{aligned} F(u, v) &= R(u, v) + jI(u, v) \\ &= |F(u, v)|e^{j\phi(u, v)} \end{aligned}$$

donde la magnitud

$$|F(u, v)| = \sqrt{R^2(u, v) + I^2(u, v)}$$

se llama *espectro de Fourier (o de frecuencia)*, y

$$\phi(u, v) = \arctan \left[\frac{I(u, v)}{R(u, v)} \right]$$

es el *ángulo de fase o espectro de fase*.

Finalmente, el espectro de potencia se define como

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

Como antes, R e I son las partes real e imaginaria de $F(u, v)$, y todos los cálculos se llevan a cabo para las variables discretas $u = 0, 1, 2, \dots, M - 1$ y $v = 0, 1, 2, \dots, N - 1$.

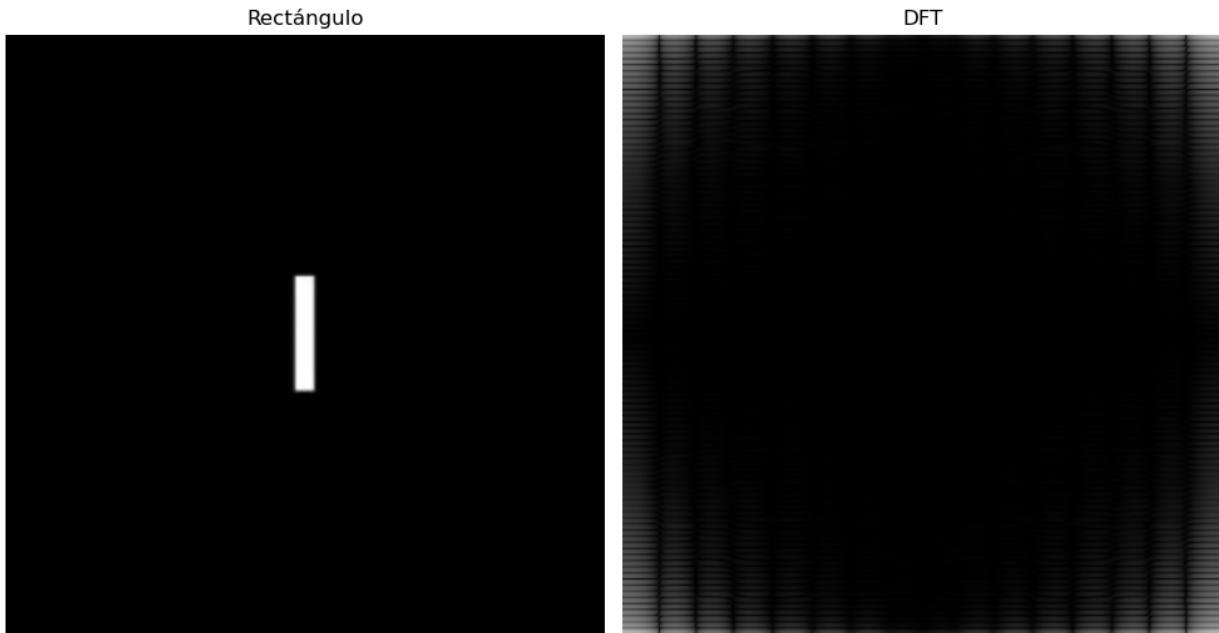
- Por lo tanto, $|F(u, v)|$, $\phi(u, v)$ y $P(u, v)$ son matrices de tamaño $M \times N$.

In [311...]

```
rect = imread('figs/rect.png', as_gray=True)
rect = resize(rect, (512, 512), anti_aliasing=True)

rect_dft = np.fft.fft2(rect)

show_images(rect, np.log(1 + np.abs(rect_dft)),
            titles=['Rectángulo', 'DFT'],
            cols=2, figsize=(10, 10))
```



Es evidente en la figura DFT que el área alrededor del origen de la transformación contiene los valores más altos (y por lo tanto aparece más brillante en la imagen).

- Sin embargo, tenga en cuenta que las cuatro esquinas del espectro contienen valores igualmente altos.
- Para centrar el espectro, simplemente multiplicamos la imagen en (a) por $(-1)^{x+y}$ antes de calcular la DFT.

In [312...]

```
# funcion para realizar shift de la imagen
def shift(img):
    """
    Realizar shift de la imagen.
    Args:
        img (np.ndarray): La imagen de entrada.
    Returns:
        np.ndarray: La imagen con shift.
    """

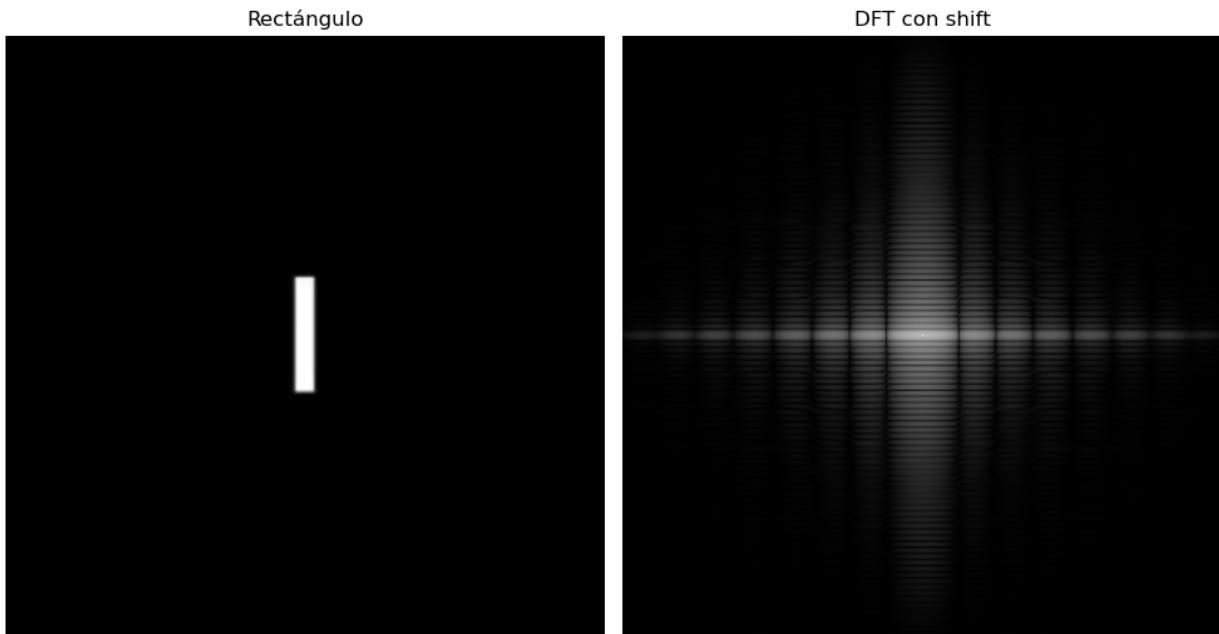
    # obtener dimensiones de la imagen
    M, N = img.shape
    # inicializar la matriz de salida
    out = np.zeros((M, N), dtype=np.complex64)
    # realizar shift
    for x in range(M):
        for y in range(N):
            out[x, y] = img[x, y] * (-1) ** (x + y)
```

```
# retornar la matriz de salida
return out
```

In [313...]

```
rect_dft_shift = np.fft.fft2(shift(rect))

show_images(rect, np.log(1 + np.abs(rect_dft_shift)),
            titles=['Rectángulo', 'DFT con shift'],
            cols=2, figsize=(10, 10))
```



La figura anterior muestra el resultado, que claramente es mucho más fácil de visualizar (obsérvese la simetría con respecto al punto central).

Para resaltar los detalles, utilizamos la transformación logarítmica definida en la ecuación en la Unidad 3 con $c = 1$.

$$s = c \log(1 + r)$$

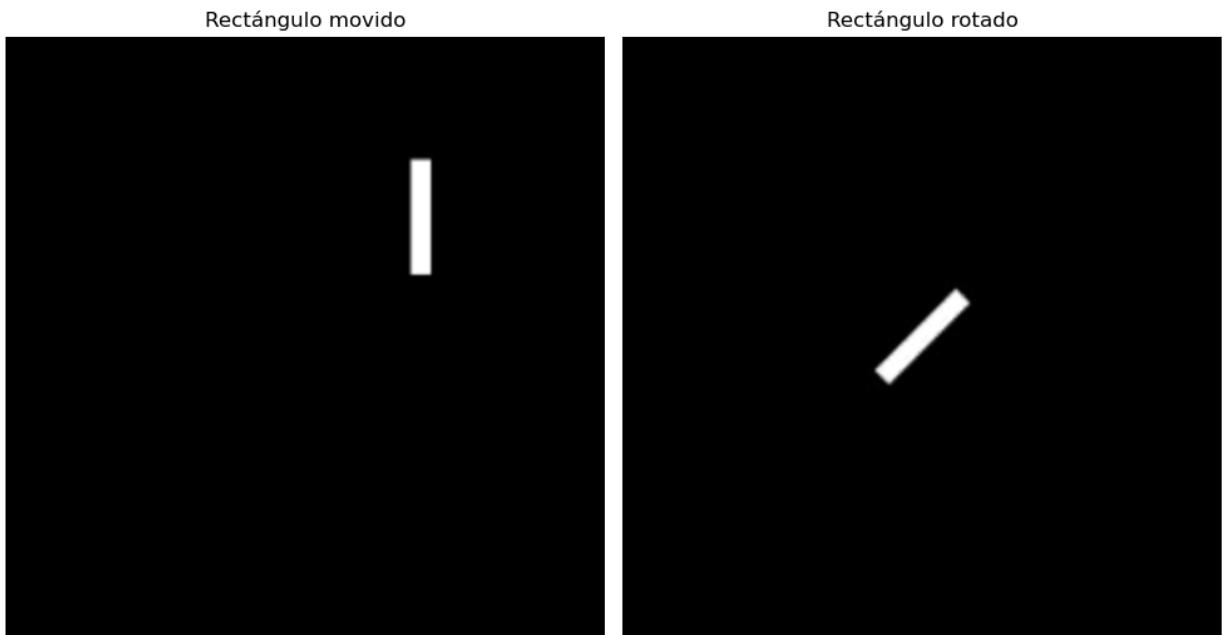
- La mayoría de los espectros que se muestran en esta unidad están escalados de esta manera.

El espectro es insensible a la traslación de la imagen (el valor absoluto del término exponencial es 1), pero gira en el mismo ángulo de una imagen rotada.

In [314...]

```
rect_movido = resize(imread('figs/rect_movido.png', as_gray=True), (512, 512))
rect_rotado = resize(imread('figs/rect_rotado.png', as_gray=True), (512, 512))

show_images(rect_movido, rect_rotado,
            titles=['Rectángulo movido', 'Rectángulo rotado'],
            cols=2, figsize=(10, 10))
```



Claramente, las imágenes de las "rect.png" y "rect_movido.png" son diferentes, por lo que, si sus espectros de Fourier son los mismos, entonces, sus ángulos de fase deben ser diferentes.

In [315...]

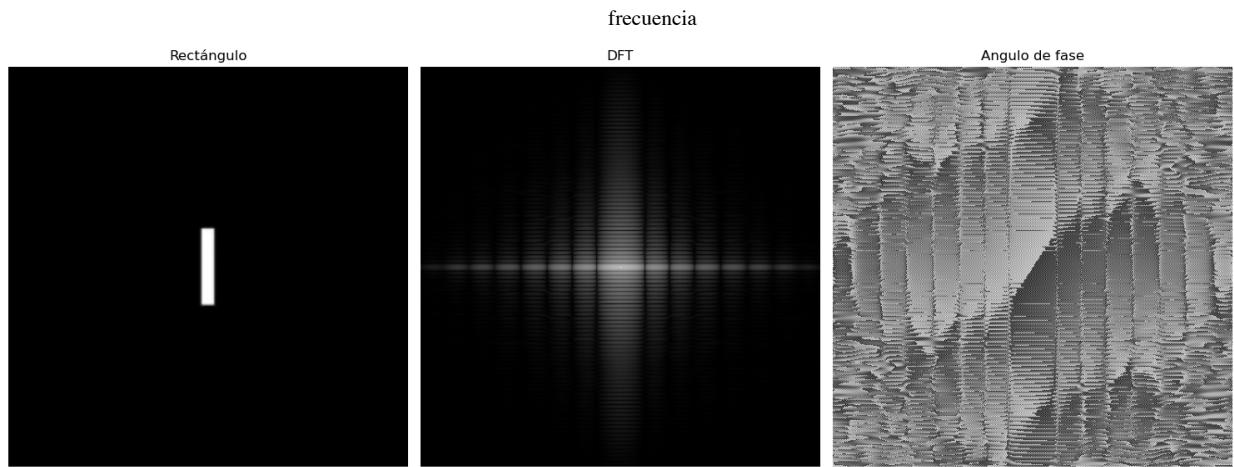
```
# funcion para calcular el angulo de fase de la imagen
def phase(img):
    """
    Calcular el angulo de fase de una imagen.
    Args:
        img (np.ndarray): La imagen de entrada.
    Returns:
        np.ndarray: El angulo de fase de la imagen.
    """
    # obtener dimensiones de la imagen
    M, N = img.shape
    # inicializar la matriz de salida
    out = np.zeros((M, N), dtype=np.float32)
    # calcular el angulo de fase
    for x in range(M):
        for y in range(N):
            out[x, y] = np.angle(img[x, y])
    # retornar la matriz de salida
    return out
```

Las figuras siguientes lo confirman.

In [316...]

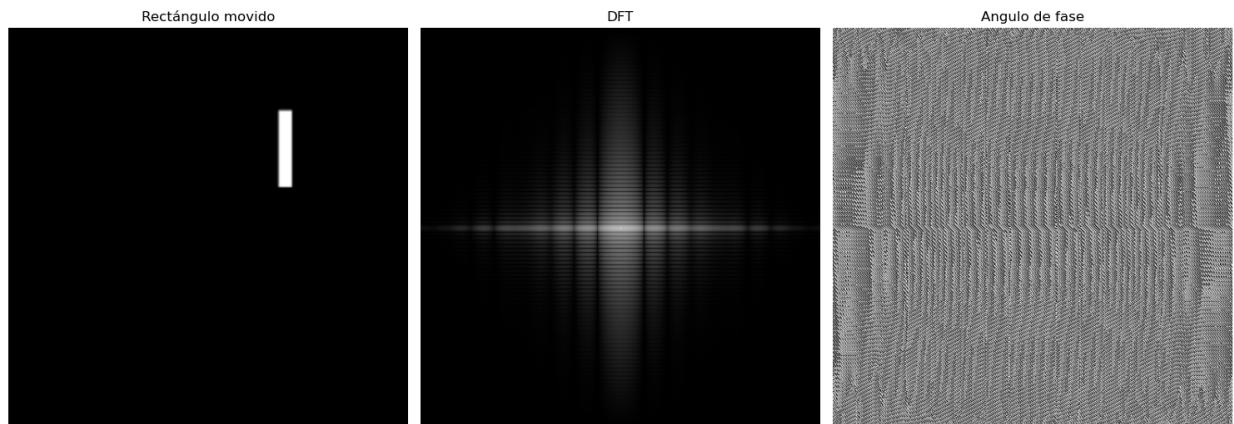
```
rect_dft_shift = np.fft.fft2(shift(rect))
rect_dft_shift_phase = phase(rect_dft_shift)

show_images(rect, np.log(1 + np.abs(rect_dft_shift)), rect_dft_shift_phase,
           titles=['Rectángulo', 'DFT', 'Angulo de fase'],
           cols=3, figsize=(15, 15))
```



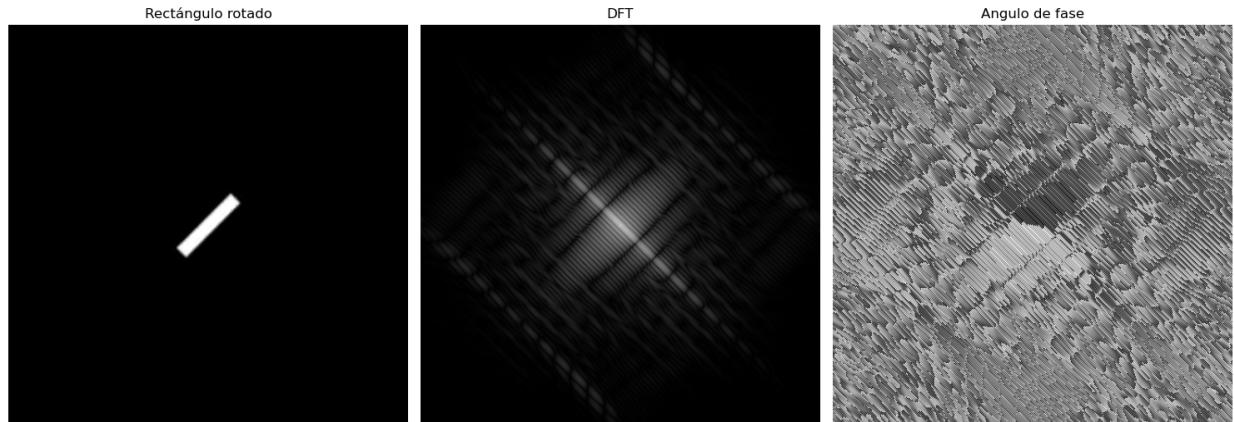
```
In [317...]: rect_movido_dft = np.fft.fft2(shift(rect_movido))
rect_movido_dft_phase = phase(rect_movido_dft)

show_images(rect_movido, np.log(1 + np.abs(rect_movido_dft)), rect_movido_dft_
           titles=['Rectángulo movido', 'DFT', 'Angulo de fase'],
           cols=3, figsize=(15, 15))
```



```
In [318...]: rect_rotado_dft = np.fft.fft2(shift(rect_rotado))
rect_rotado_dft_phase = phase(rect_rotado_dft)

show_images(rect_rotado, np.log(1 + np.abs(rect_rotado_dft)), rect_rotado_dft_
           titles=['Rectángulo rotado', 'DFT', 'Angulo de fase'],
           cols=3, figsize=(15, 15))
```



```
In [319...]: lena_dft_shift = np.fft.fft2(shift(lena))
```

```
show_images(lena, np.log(1 + np.abs(lena_dft_shift)),
            titles=['Imagen original', 'DFT con shift'],
            cols=2, figsize=(10, 10))
```



4.6.2. El teorema de convolución 2D discreta

Ampliando la convolución a dos variables da como resultado la siguiente expresión para convolución circular 2-D:

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

para $x = 0, 1, 2, \dots, M - 1$ y $y = 0, 1, 2, \dots, N - 1$. Como en la ecuación. (4-48), ecuación. (4-94) da un período de una secuencia periódica 2-D.

El teorema de convolución 2-D viene dado por:

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

y por el contrario,

$$(f \cdot h)(x, y) \Leftrightarrow \frac{1}{MN} (F * H)(u, v)$$

donde F y H son las transformadas de Fourier de f y h , respectivamente.

Como antes, la flecha doble se utiliza para indicar que los lados izquierdo y derecho de las expresiones constituyen un par de transformada de Fourier.

- Nuestro interés en el resto de esta unidad está en la ecuación, que establece que la transformada de Fourier de la convolución espacial de f y h , es el producto de sus transformadas.

- De manera similar, la DFT inversa del producto $(F \cdot H)(u, v)$ produce $(f \star h)(x, y)$.

	Descripción	Expresiones
1	Transformada discreta de Fourier (DFT) de $f(x, y)$	$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$
2	Transformada discreta de Fourier inversa (IDFT) de $F(u, v)$	$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$
3	Espectro	$ F(u, v) = [R^2(u, v) + I^2(u, v)]^{1/2}$
4	Ángulo de fase	$\phi(u, v) = \tan^{-1} \left[\frac{I(u, v)}{R(u, v)} \right]$
5	Representación polar	$F(u, v) = F(u, v) e^{j\phi(u, v)}$
6	Espectro de poder	$P(u, v) = F(u, v) ^2$
	Descripción	Expresiones
7	Valor promedio	$\bar{f} = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) = \frac{1}{MN} \int \int f(x, y) dx dy$
8	Periodicidad (k_1 y k_2 son enteros)	$F(u, v) = F(u + k_1 M, v) = F(u, v + k_2 N)$ $= F(u + k_1, v + k_2 N)$ $f(x, y) = f(x + k_1 M, y) = f(x, y + k_2 N)$ $= f(x + k_1 M, y + k_2 N)$
9	Convolución	$(f \star h)(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) h(x - m, y - n)$
10	Correlación	$(f * h)(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f^*(m, n) h(x - m, y - n)$

4.7. Bases para filtrado en el dominio de la frecuencia

4.7.1. Fundamentos

4.7.2. Pasos para el filtrado en el dominio de la frecuencia

El proceso de filtrado en el dominio de la frecuencia se puede resumir de la siguiente manera:

1. Dada una imagen de entrada $f(x, y)$ de tamaño $M \times N$, obtenga los tamaños de relleno $P = 2M$ y $Q = 2N$.
2. A partir de la imagen rellenada $f_p(x, y)$ de tamaño $P \times Q$ usando relleno cero, espejo o réplica.

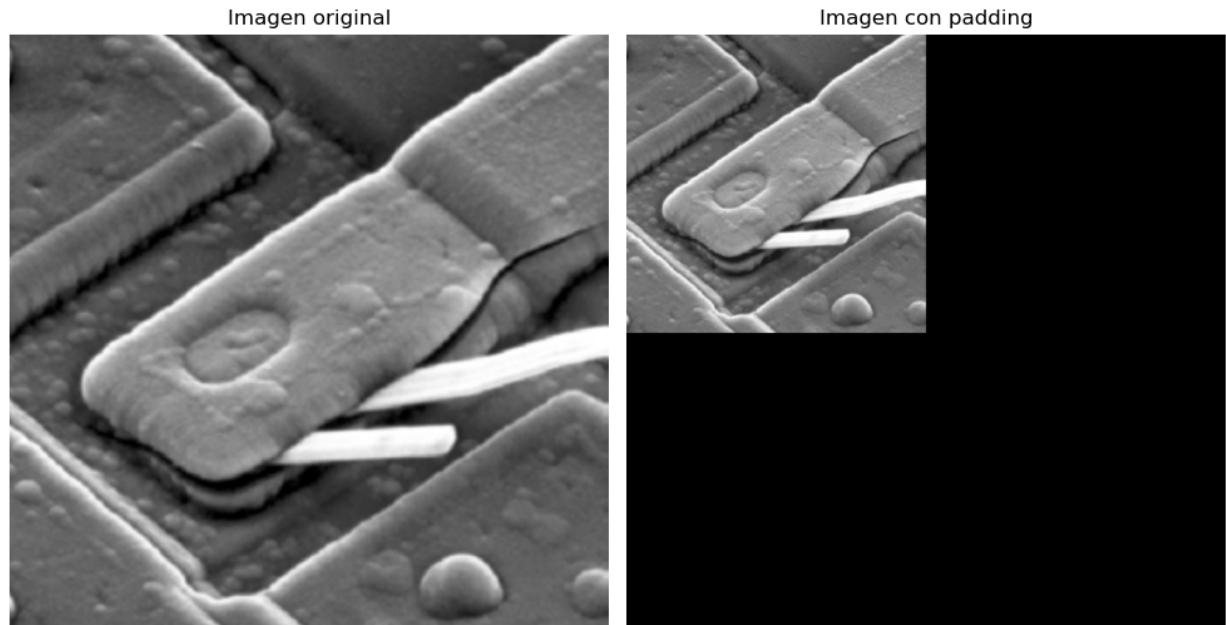
In [320...]

```
size = 512
f = resize(imread('figs/electronic.png', as_gray=True), (size, size))
f_p = np.pad(f, ((0, size), (0, size)), 'constant')

P, Q = f_p.shape

show_images(f, f_p,
            titles=['Imagen original', 'Imagen con padding'],
            cols=2, figsize=(10, 10))

print(P, Q)
```

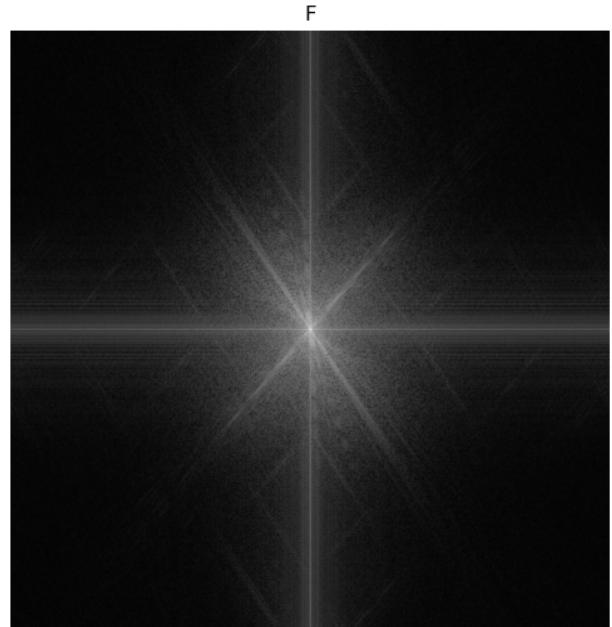
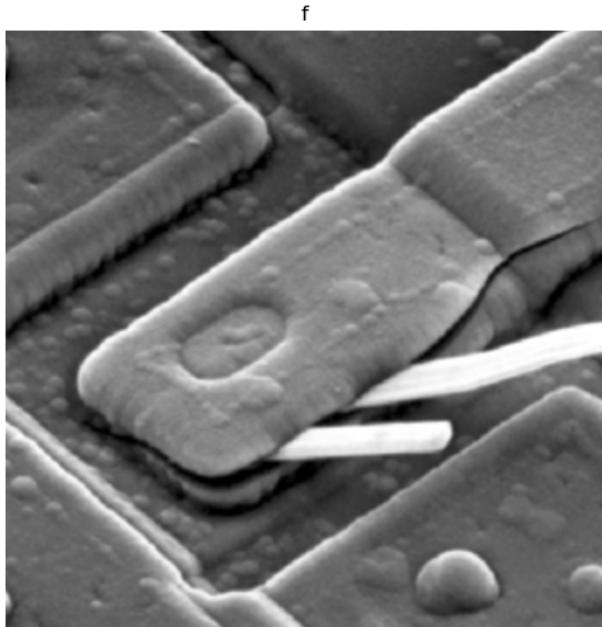


1024 1024

1. Multiplique $f_p(x, y)$ por $(-1)^{x+y}$ para centrar la transformada de Fourier en el rectángulo de frecuencia $P \times Q$.
2. Calcule el DFT, $F(u, v)$, de la imagen del Paso 3.

In [321...]: `F = np.fft.fft2(shift(f_p))`

```
show_images(f, np.log(1 + np.abs(F)),
            titles=['f', 'F'],
            cols=2, figsize=(10, 10))
```



1. Construya una función de transferencia de filtro simétrica real, $H(u, v)$, de tamaño $P \times Q$ con centro en $(P/2, Q/2)$.

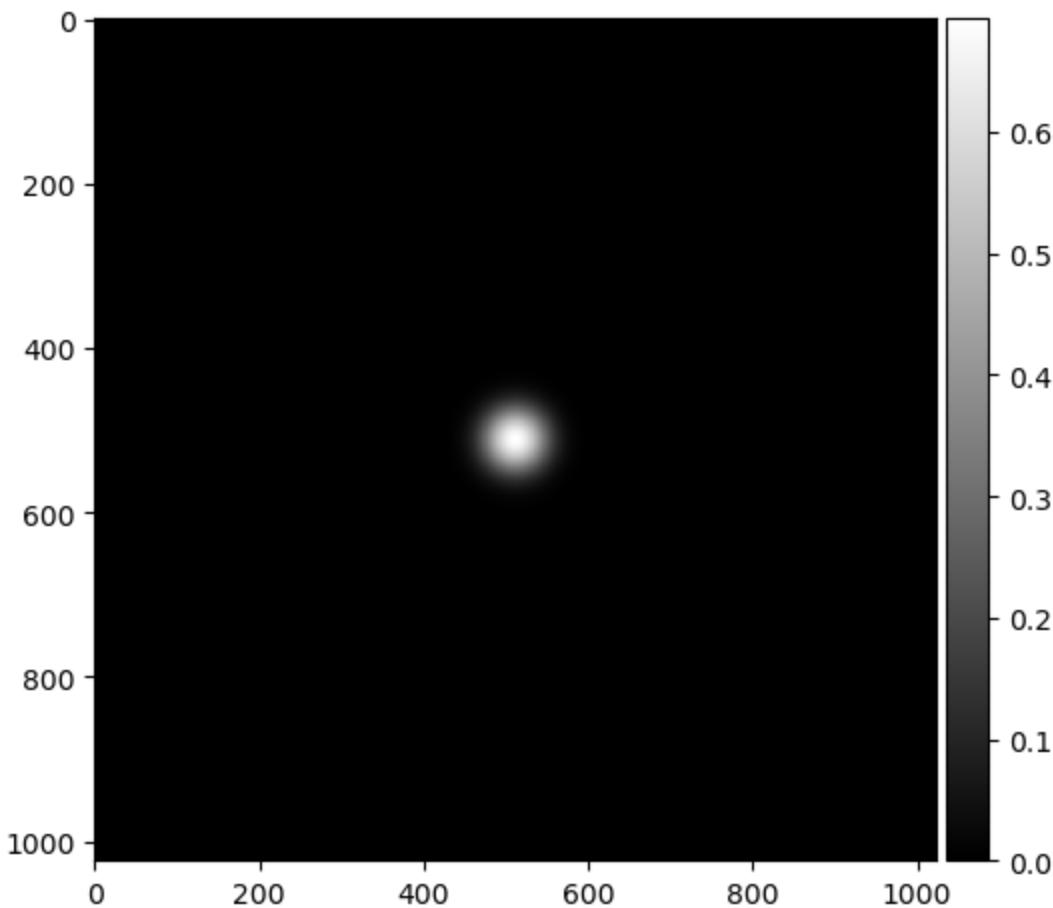
In [322...]:

```
# crear funcion de transferencia H gaussiano
def GaussianLowPassFilter(P, Q, sigma=10):
    """
    Crear la funcion de transferencia H gaussiano.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        sigma (float): El valor de sigma (default=10).
    Returns:
        np.ndarray: La funcion de transferencia H gaussiano.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            out[u, v] = np.exp(-((u - P / 2) ** 2 + (v - Q / 2) ** 2) / (2 * sigma))
    # retornar la matriz de salida
    return out
```

In [323...]:

```
H = GaussianLowPassFilter(P, Q, 25)
imshow(np.log(1 + np.abs(H)), cmap='gray')
```

Out[323]: <matplotlib.image.AxesImage at 0x13e1bcd0>

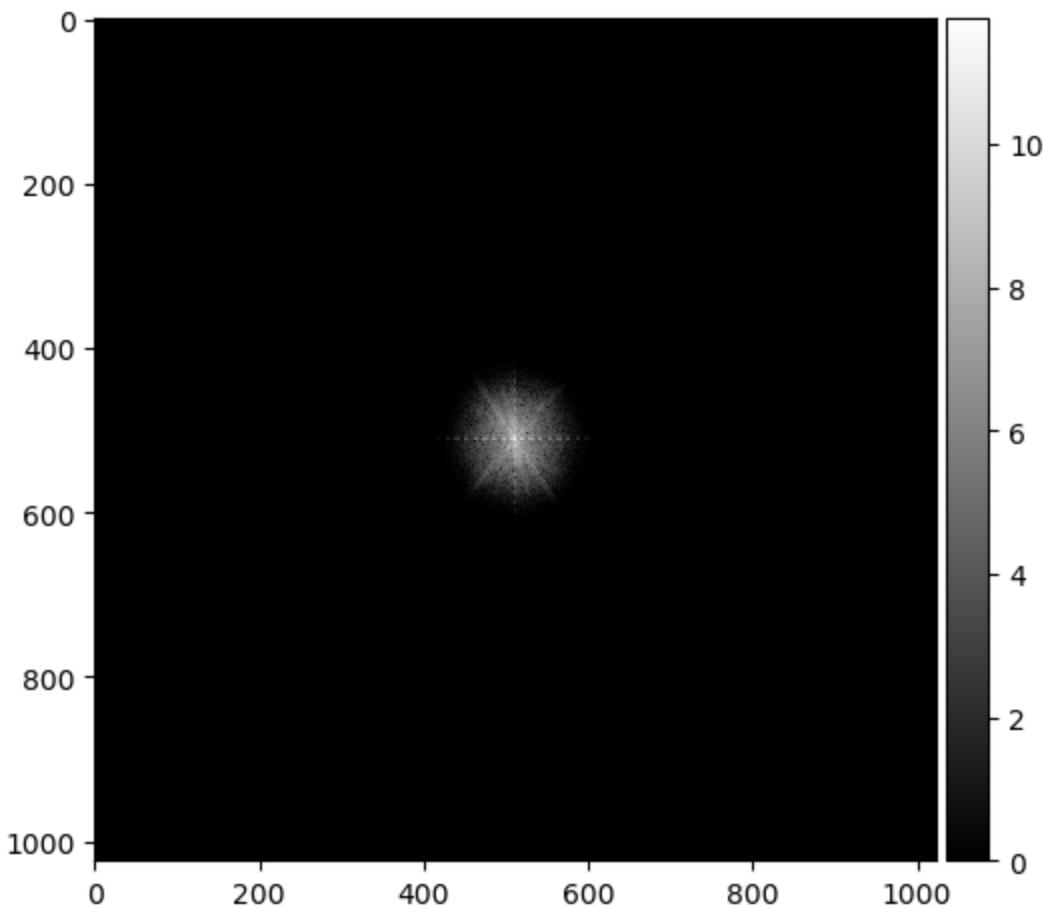


1. Forme el producto $G(u, v) = H(u, v)F(u, v)$ usando la multiplicación por elementos; es decir, $G(i, k) = H(i, k)F(i, k)$ para $i = 0, 1, 2, \dots, M - 1$ y $k = 0, 1, 2, \dots, N - 1$.

In [324...]

```
G = F * H  
imshow(np.log(1 + np.abs(G)), cmap='gray')
```

Out[324]: <matplotlib.image.AxesImage at 0x13daab110>



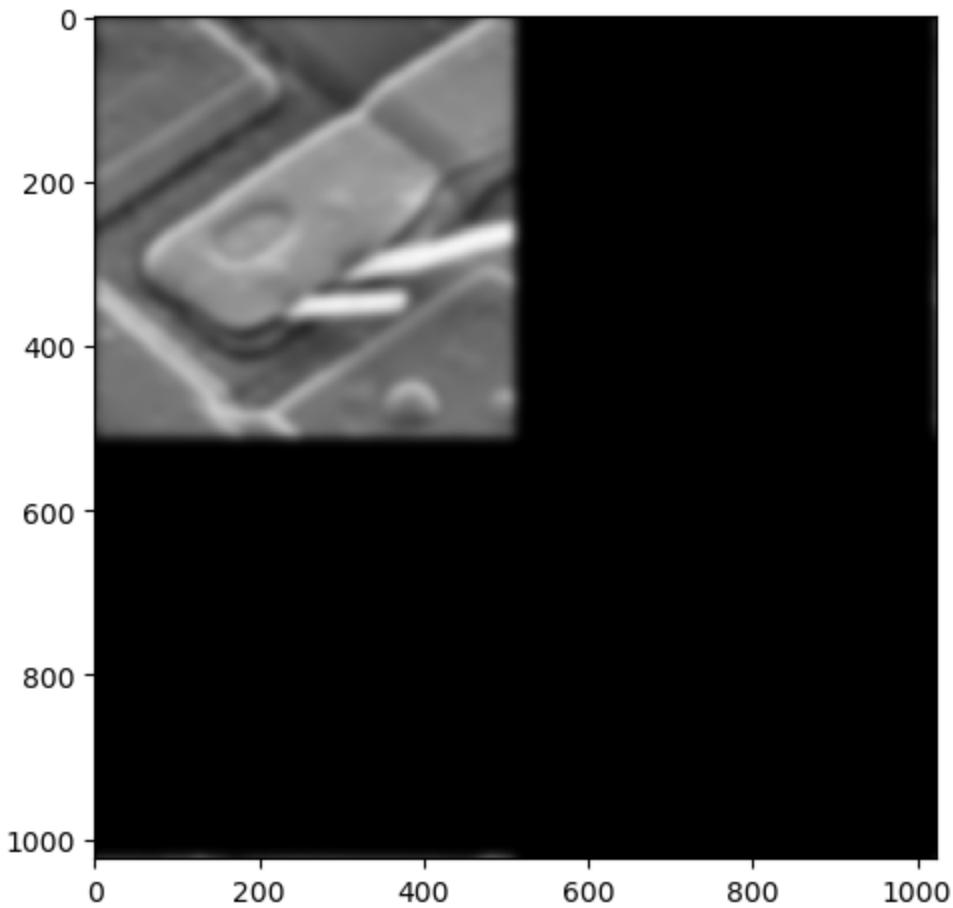
1. Obtenga la imagen filtrada (de tamaño $P \times Q$) calculando el IDFT de $G(u, v)$:

$$g_p(x, y) = (\text{real}[\mathcal{J}^{-1}\{G(u, v)\}]) (-1)^{x+y}$$

In [325]: `g_p = np.fft.ifft2(G)`

```
imshow(np.abs(g_p), cmap='gray')
```

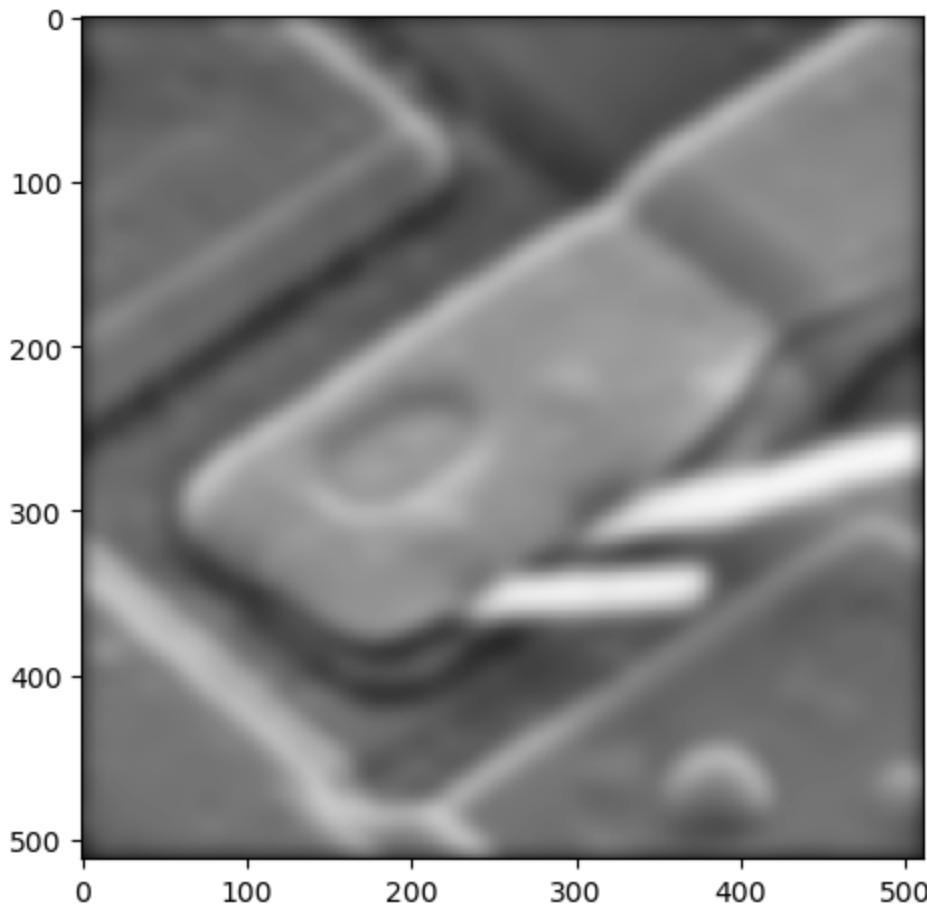
Out[325]: <matplotlib.image.AxesImage at 0x13e33c610>



1. Obtenga el resultado final filtrado, $g(x, y)$, del mismo tamaño que la imagen de entrada, extrayendo la región $M \times N$ del cuadrante superior izquierdo de $g_p(x, y)$.

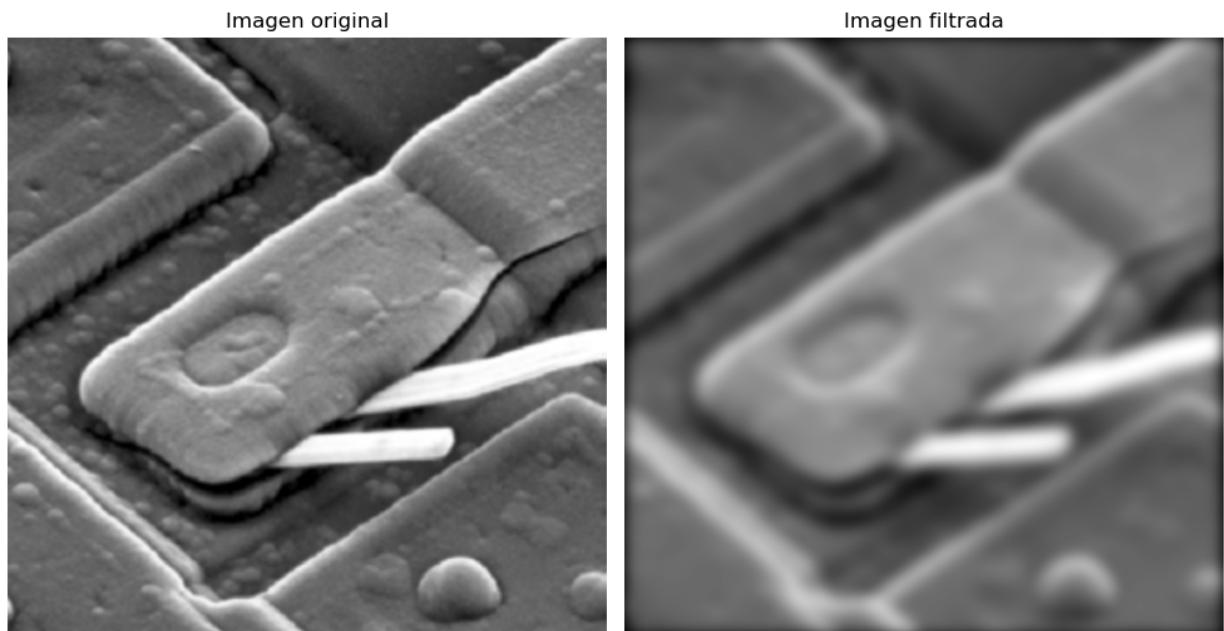
```
In [326]: g = g_p[:size, :size]
imshow(np.abs(g), cmap='gray')
```

```
Out[326]: <matplotlib.image.AxesImage at 0x13e484250>
```



In [327...]

```
show_images(f, np.abs(g),
            titles=['Imagen original', 'Imagen filtrada'],
            cols=2, figsize=(10, 10))
```



In [328...]

```
def filtro_frecuencia(img, filter, padding='reflect'):
    """
    Filtrar una imagen en el dominio de la frecuencia.
    Args:
        img (np.ndarray): La imagen de entrada.
```

```

        filter (np.ndarray): La funcion de transferencia.
    Returns:
        np.ndarray: La imagen filtrada.
    .....
    # obtener dimensiones de la imagen
    M, N = img.shape
    # realizar padding
    img_p = np.pad(img, ((0, M), (0, N)), padding)
    # calcular la transformada de Fourier de la imagen
    F = np.fft.fft2(shift(img_p))
    # calcular la funcion de transferencia
    H = filter
    # calcular la transformada filtrada
    G = F * H
    # calcular la transformada inversa
    g_p = np.fft.ifft2(G)
    # obtener la imagen filtrada
    g = g_p[:M, :N]
    # retornar la imagen filtrada
    return np.abs(g)

```

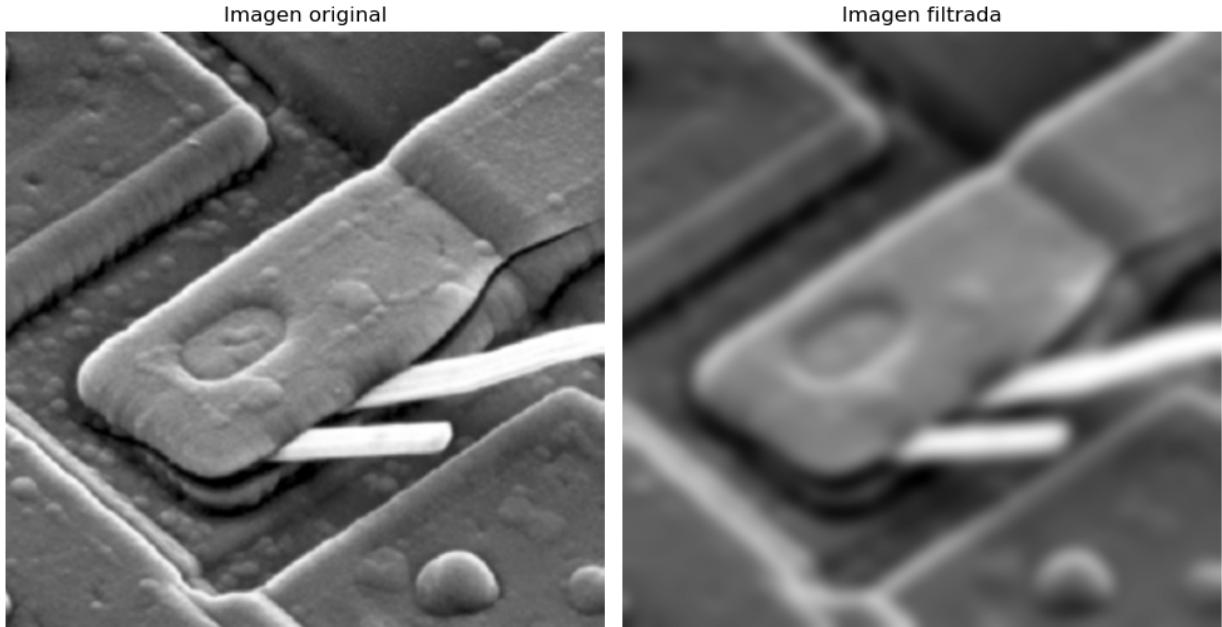
In [329...]

```

filtro = GaussianLowPassFilter(P, Q, 25)
res = filtro_frecuencia(f, filtro, padding='reflect')

show_images(f, res,
            titles=['Imagen original', 'Imagen filtrada'],
            cols=2, figsize=(10, 10))

```



4.8. Filtros pasa bajas en el dominio de la frecuencia

En esta sección, consideramos tres tipos de filtros de paso bajo: ideal, Butterworth y Gaussiano.

Estas tres categorías cubren el rango desde un filtrado muy nítido (ideal) hasta un filtrado muy suave (gaussiano).

La forma de un filtro Butterworth está controlada por un parámetro llamado orden de filtro.

- Para valores grandes de este parámetro, el filtro Butterworth se acerca al filtro ideal.
- Para valores más bajos, el filtro Butterworth se parece más a un filtro gaussiano.

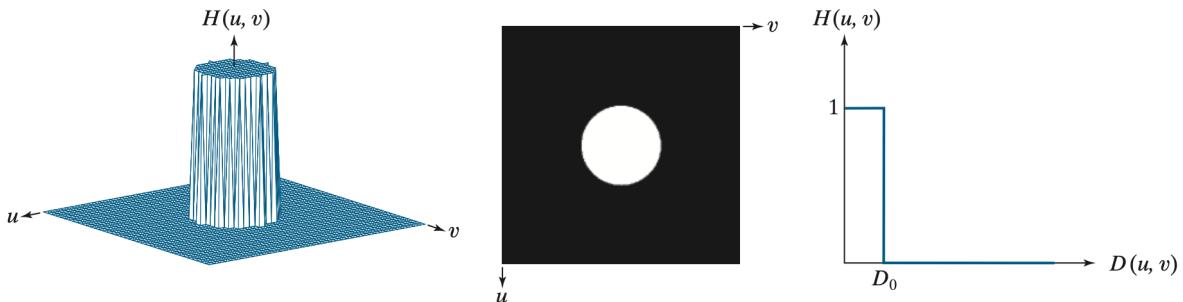
4.8.1. Filtro pasa bajas ideal

Un filtro de paso bajo 2-D que pasa sin atenuación todas las frecuencias dentro de un círculo de radio desde el origen y "corta" todas las frecuencias fuera de este círculo se llama filtro de paso bajo ideal (ILPF); está especificado por la función de transferencia

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

donde D_0 es una constante positiva y $D(u, v)$ es la distancia entre un punto (u, v) en el dominio de la frecuencia y el centro del rectángulo de frecuencia $P \times Q$; eso es,

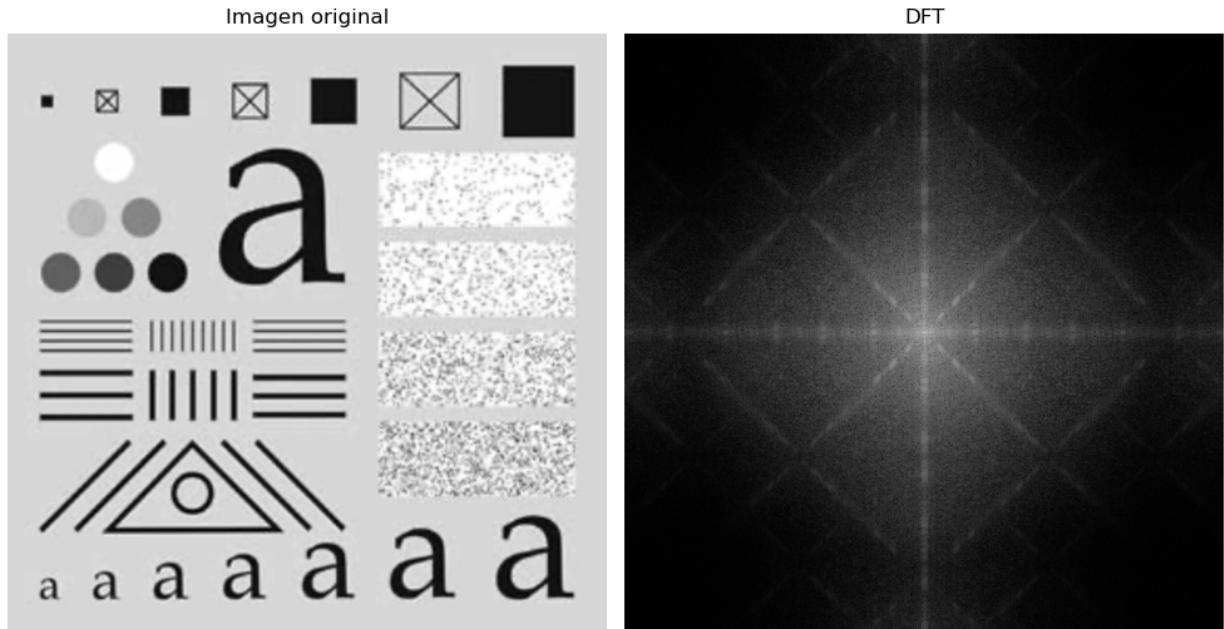
$$D(u, v) = [(u - P/2)^2 + (v - Q/2)^2]^{1/2}$$



```
In [330...]: test = resize(imread('figs/test_pattern.png', as_gray=True), (688, 688))
test_dft = np.fft.fft2(shift(test))

P = 2 * test.shape[0]
Q = 2 * test.shape[1]

show_images(test, np.log(1 + np.abs(test_dft)),
            titles=['Imagen original', 'DFT'],
            cols=2, figsize=(10, 10))
```

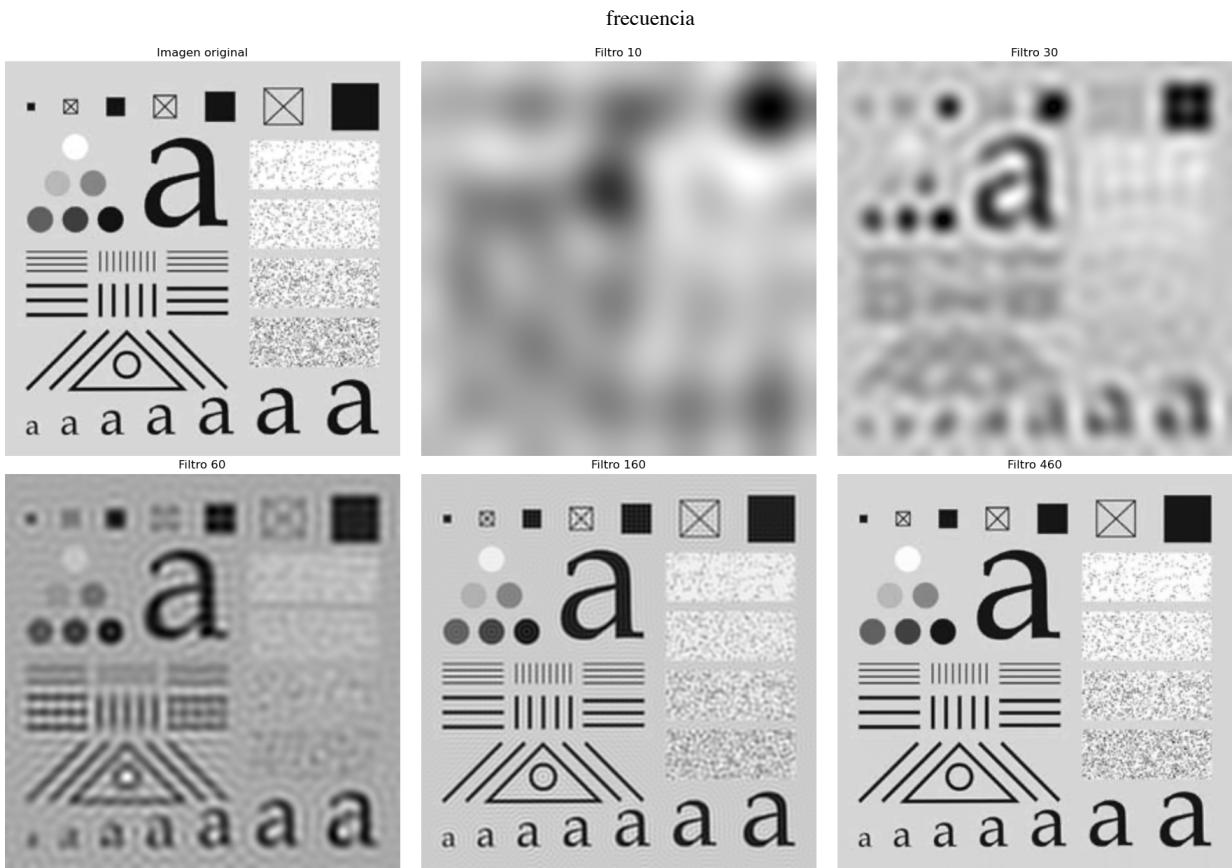


```
In [331...]: def IdealLowPassFilter(P, Q, rad):
    """
    Crear la funcion de transferencia Ideal LPF.
    Args:
        rad (int): El radio del filtro.
    Returns:
        np.ndarray: La funcion de transferencia ILPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            if D <= rad:
                out[u, v] = 1
    # retornar la matriz de salida
    return out
```

```
In [332...]: HI_10 = IdealLowPassFilter(P, Q, 10)
HI_30 = IdealLowPassFilter(P, Q, 30)
HI_60 = IdealLowPassFilter(P, Q, 60)
HI_160 = IdealLowPassFilter(P, Q, 160)
HI_460 = IdealLowPassFilter(P, Q, 460)
```

```
In [333...]: test_I10 = filtro_frecuencia(test, HI_10)
test_I30 = filtro_frecuencia(test, HI_30)
test_I60 = filtro_frecuencia(test, HI_60)
test_I160 = filtro_frecuencia(test, HI_160)
test_I460 = filtro_frecuencia(test, HI_460)
```

```
In [334...]: show_images(test, test_I10, test_I30, test_I60, test_I160, test_I460,
               titles=['Imagen original', 'Filtro 10', 'Filtro 30', 'Filtro 60',
                       'Filtro 160', 'Filtro 460'],
               cols=3, figsize=(18, 12))
```



4.8.2. Filtro pasa bajas Gaussiano

Las funciones de transferencia del filtro de paso bajo gaussiano (GLPF) tienen la forma

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

donde $D(u, v)$ es la distancia desde el centro del rectángulo de frecuencia $P \times Q$ hasta cualquier punto, (u, v) , contenido por el rectángulo.

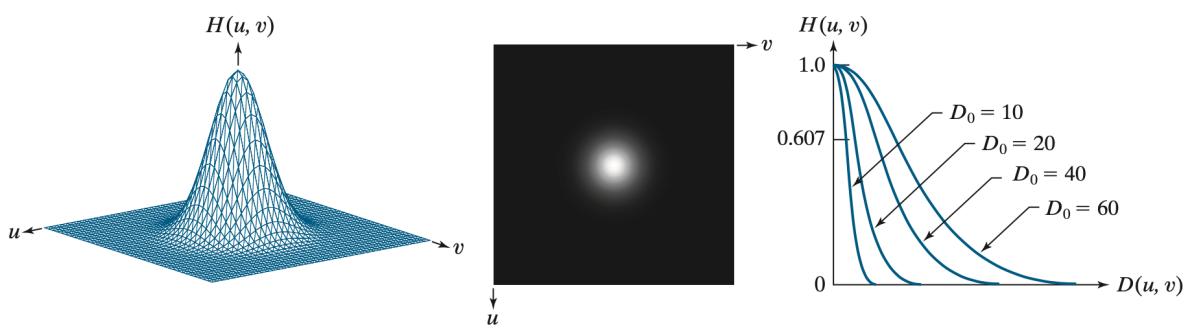
A diferencia de expresiones anteriores para funciones gaussianas, aquí no utilizamos una constante multiplicadora para ser coherentes con los filtros analizados en esta unidad , cuyo valor más alto es 1.

Como antes, σ es una medida de dispersión alrededor del centro.

- Al dejar $\sigma = D_0$, podemos expresar la función de transferencia gaussiana en la misma notación que otras funciones en esta sección:

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

donde D_0 es la frecuencia de corte. Cuando $D(u, v) = D_0$, la función de transferencia GLPF se reduce a 0.607 de su valor máximo de 1.0.

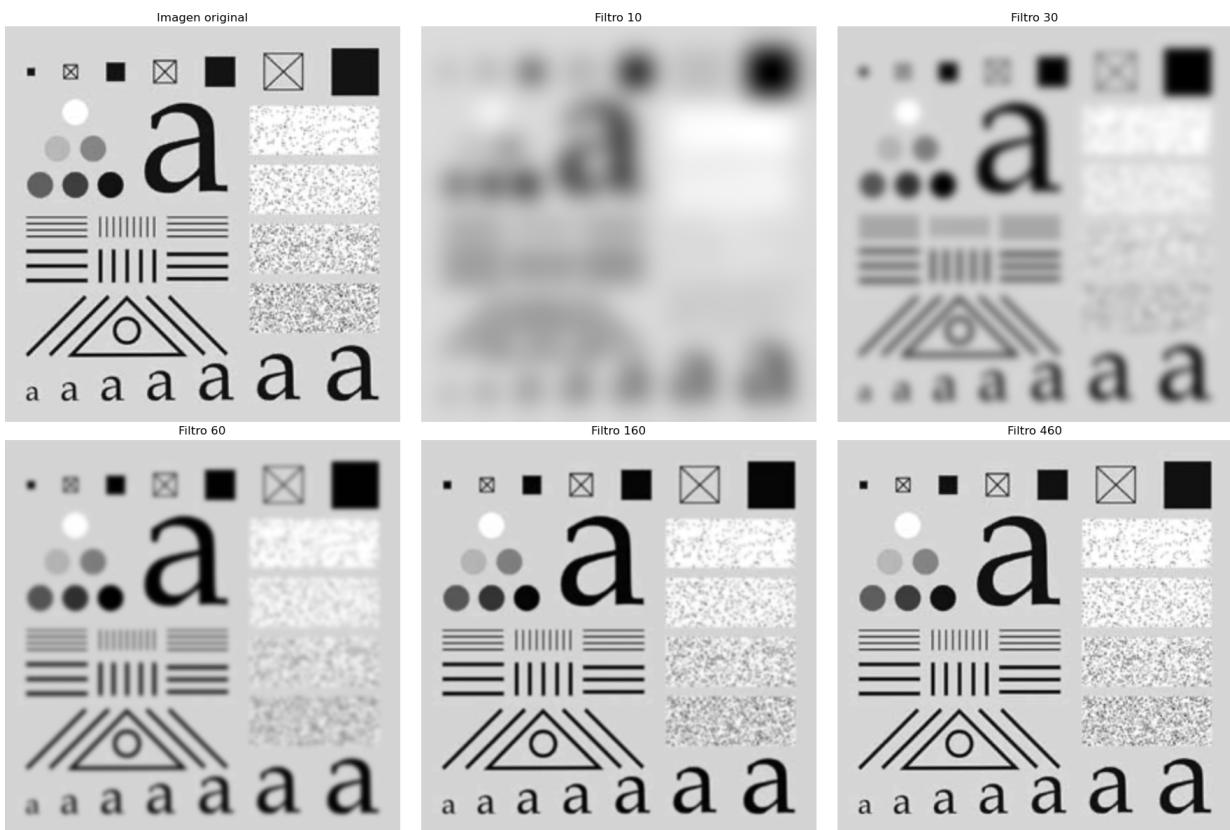


```
In [335...]: HG_10 = GaussianLowPassFilter(P, Q, 10)
HG_30 = GaussianLowPassFilter(P, Q, 30)
HG_60 = GaussianLowPassFilter(P, Q, 60)
HG_160 = GaussianLowPassFilter(P, Q, 160)

HG_460 = GaussianLowPassFilter(P, Q, 460)
```

```
In [336...]: test_G10 = filtro_frecuencia(test, HG_10)
test_G30 = filtro_frecuencia(test, HG_30)
test_G60 = filtro_frecuencia(test, HG_60)
test_G160 = filtro_frecuencia(test, HG_160)
test_G460 = filtro_frecuencia(test, HG_460)
```

```
In [337...]: show_images(test, test_G10, test_G30, test_G60, test_G160, test_G460,
titles=['Imagen original', 'Filtro 10', 'Filtro 30', 'Filtro 60',
'Filtro 160', 'Filtro 460'],
cols=3, figsize=(18, 12))
```



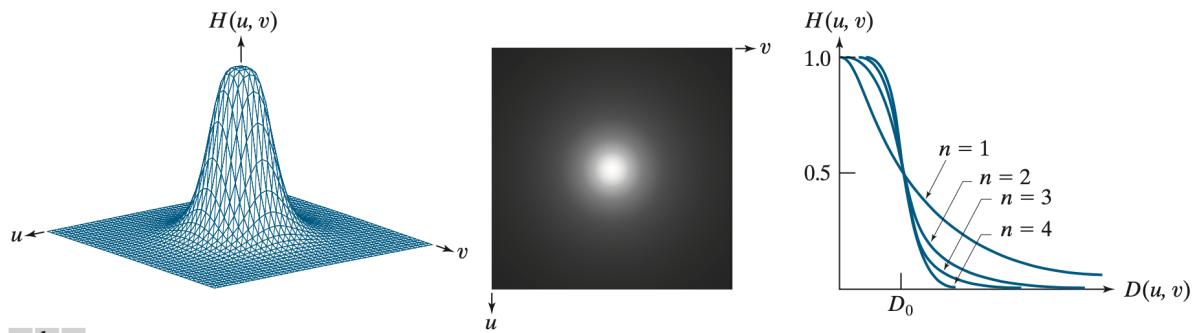
4.8.3. Filtro pasa bajas de Butterworth

La función de transferencia de un filtro de paso bajo Butterworth (BLPF) de orden n , con frecuencia de corte a una distancia D_0 del centro del rectángulo de frecuencia, se define como

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

La Figura muestra un gráfico en perspectiva, visualización de imágenes y secciones transversales radiales de la función BLPF.

Comparando los gráficos de sección transversal anteriores, vemos que la función BLPF se puede controlar para aproximarse a las características del ILPF usando valores más altos de n y del GLPF para valores más bajos de n , al tiempo que proporciona una transición suave desde niveles bajos a altas frecuencias.



In [338]:

```
def ButterworthLowPassFilter(P, Q, rad, n):
    """
    Crear la función de transferencia Butterworth LPF.
    Args:
        rad (int): El radio del filtro.
        n (int): El orden del filtro.
    Returns:
        np.ndarray: La función de transferencia BLPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la función de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            out[u, v] = 1 / (1 + (D / rad) ** (2 * n))
    # retornar la matriz de salida
    return out
```

In [339]:

```
HB_10 = ButterworthLowPassFilter(P, Q, 10, 2.25)
HB_30 = ButterworthLowPassFilter(P, Q, 30, 2.25)
HB_60 = ButterworthLowPassFilter(P, Q, 60, 2.25)
HB_160 = ButterworthLowPassFilter(P, Q, 160, 2.25)
HB_460 = ButterworthLowPassFilter(P, Q, 460, 2.25)
```

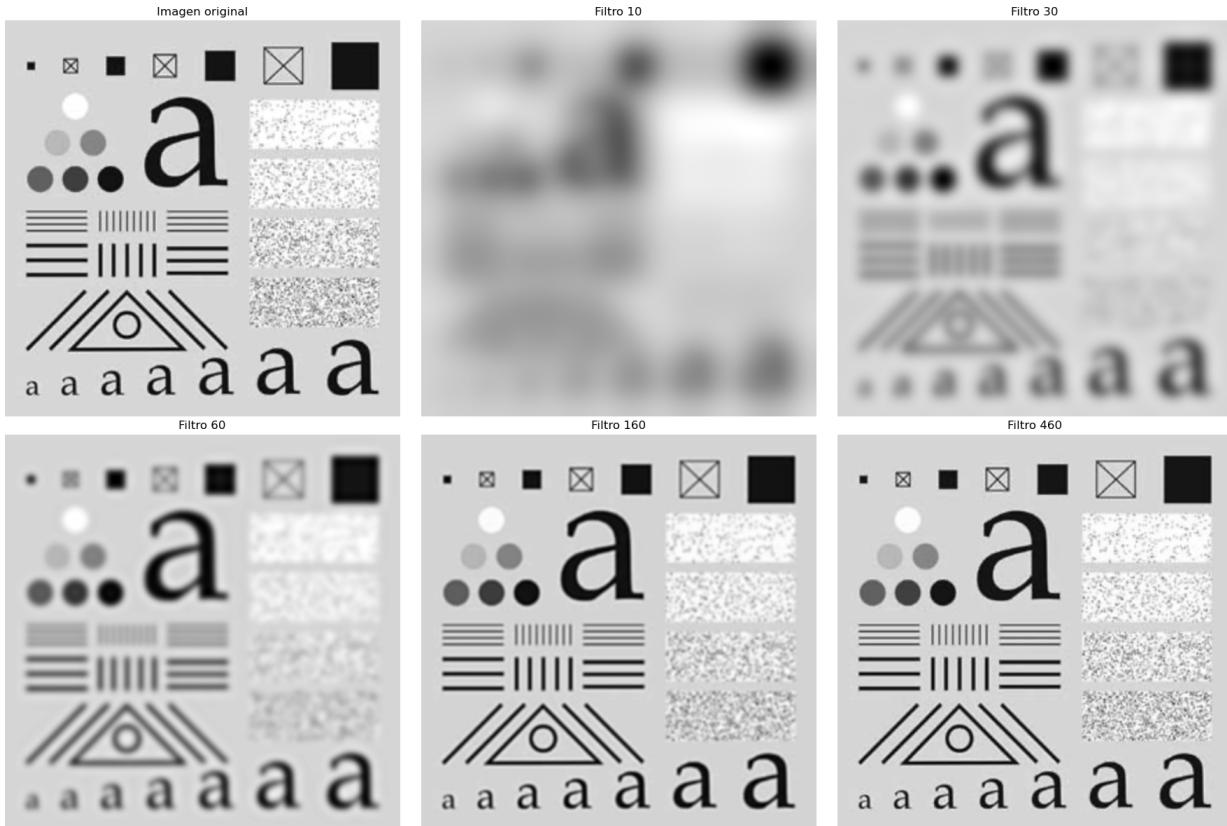
In [340]:

```
test_B10 = filtro_frecuencia(test, HB_10)
test_B30 = filtro_frecuencia(test, HB_30)
```

```
test_B60 = filtro_frecuencia(test, HB_60)
test_B160 = filtro_frecuencia(test, HB_160)
test_B460 = filtro_frecuencia(test, HB_460)
```

In [341...]

```
show_images(test, test_B10, test_B30, test_B60, test_B160, test_B460,
           titles=['Imagen original', 'Filtro 10', 'Filtro 30', 'Filtro 60',
                   'Filtro 160', 'Filtro 460'],
           cols=3, figsize=(18, 12))
```



In [342...]

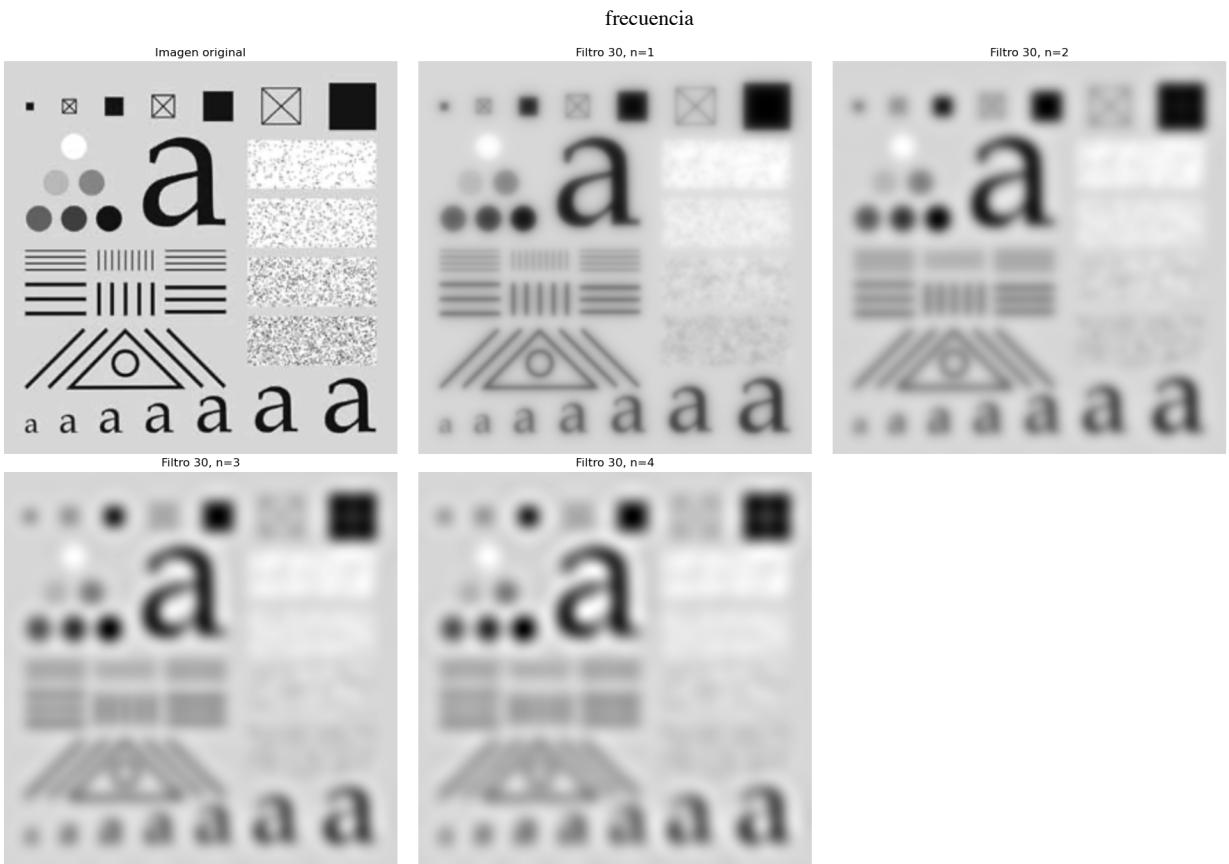
```
HB_30_1 = ButterworthLowPassFilter(P, Q, 30, 1)
HB_30_2 = ButterworthLowPassFilter(P, Q, 30, 2)
HB_30_3 = ButterworthLowPassFilter(P, Q, 30, 3)
HB_30_4 = ButterworthLowPassFilter(P, Q, 30, 4)
```

In [343...]

```
test_B30_1 = filtro_frecuencia(test, HB_30_1)
test_B30_2 = filtro_frecuencia(test, HB_30_2)
test_B30_3 = filtro_frecuencia(test, HB_30_3)
test_B30_4 = filtro_frecuencia(test, HB_30_4)
```

In [344...]

```
show_images(test, test_B30_1, test_B30_2, test_B30_3, test_B30_4,
           titles=['Imagen original', 'Filtro 30, n=1', 'Filtro 30, n=2',
                   'Filtro 30, n=3', 'Filtro 30, n=4'],
           cols=3, figsize=(18, 12))
```



4.9. Filtros pasa altas en el dominio de la frecuencia

En la sección anterior mostramos que una imagen se puede suavizar atenuando los componentes de alta frecuencia de su transformada de Fourier.

Debido a que los bordes y otros cambios abruptos en las intensidades están asociados con componentes de alta frecuencia, la nitidez de la imagen se puede lograr en el dominio de la frecuencia mediante filtrado de paso alto, que atenúa los componentes de baja frecuencia sin perturbar las altas frecuencias en la transformada de Fourier.

4.9.1. Filtros pasa altas ideal, Gaussiano y de Butterworth, a partir de filtros pasa bajas

Como fue el caso con los núcleos en el dominio espacial, restar de 1 una función de transferencia de filtro de paso bajo produce la función de transferencia de filtro de paso alto correspondiente en el dominio de frecuencia:

$$H_{HP}(u, v) = 1 - H_{LP}(u, v)$$

donde $H_{LP}(u, v)$ es la función de transferencia de un filtro de paso bajo.

Por lo tanto, se deduce que una función de transferencia de filtro de paso alto ideal (IHPF) está dada por

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

donde, como antes, $D(u, v)$ es la distancia desde el centro del rectángulo de frecuencia $P \times Q$.

De manera similar, se deduce que la función de transferencia de una función de transferencia de filtro de paso alto gaussiano (GHPF) está dada por

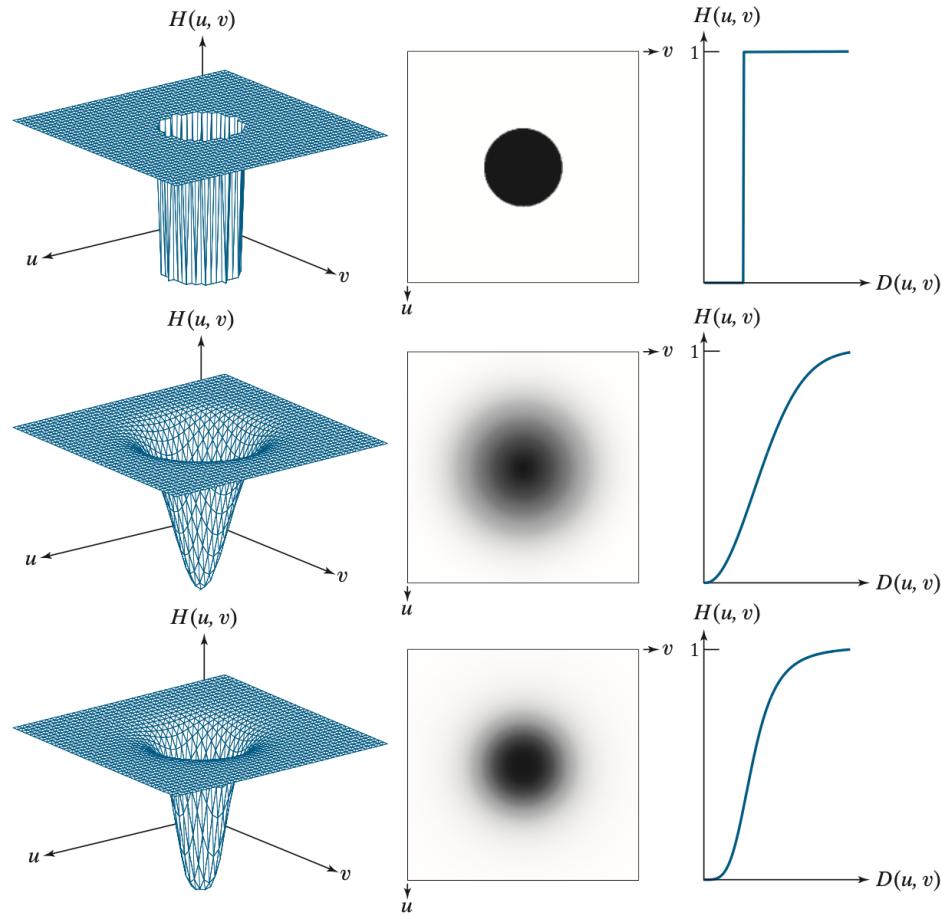
$$H(u, v) = 1 - e^{-D^2(u, v)/2D_0^2}$$

También podemos deducir que la función de transferencia de un filtro de paso alto Butterworth (BHPF) es

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

La figura muestra gráficos tridimensionales, representaciones de imágenes y secciones transversales radiales para las funciones de transferencia anteriores.

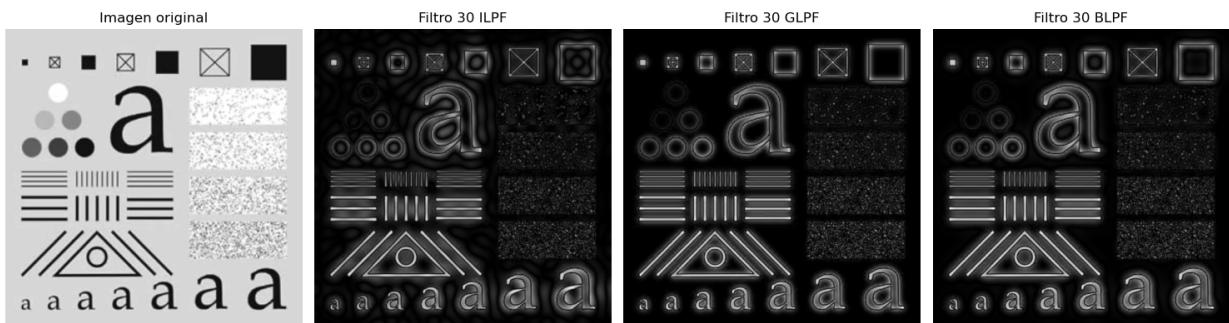
Como antes, vemos que la función de transferencia BHPF en la tercera fila de la figura representa una transición entre la nitidez del IHPF y la suavidad general de la función de transferencia GHPF.



```
In [345...]: HPI_30 = 1 - HI_30
HPG_30 = 1 - HG_30
HPB_30 = 1 - HB_30
```

```
In [346...]: test_HPI30 = filtro_frecuencia(test, HPI_30)
test_HPG30 = filtro_frecuencia(test, HPG_30)
test_HPB30 = filtro_frecuencia(test, HPB_30)
```

```
In [347...]: show_images(test, test_HPI30, test_HPG30, test_HPB30,
               titles=['Imagen original', 'Filtro 30 ILPF', 'Filtro 30 GLPF',
                       'Filtro 30 BLPF'],
               cols=4, figsize=(15, 15))
```



```
In [348...]: def IdealHighPassFilter(P, Q, rad):
    """
    Crear la funcion de transferencia Ideal HPF.
    Args:
        rad (int): El radio del filtro.
    Returns:
        np.ndarray: La funcion de transferencia IHPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            if D >= rad:
                out[u, v] = 1
    # retornar la matriz de salida
    return out
```

```
In [349...]: def GaussianHighPassFilter(P, Q, sigma):
    """
    Crear la funcion de transferencia Gaussian HPF.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        sigma (float): El valor de sigma.
    Returns:
        np.ndarray: La funcion de transferencia GHPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
```

```

        D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
        out[u, v] = 1 - np.exp(-D ** 2 / (2 * sigma ** 2))
    # retornar la matriz de salida
    return out

```

In [350...]:

```

def ButterworthHighPassFilter(P, Q, rad, n):
    """
    Crear la funcion de transferencia Butterworth HPF.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        rad (int): El radio del filtro.
        n (int): El orden del filtro.
    Returns:
        np.ndarray: La funcion de transferencia BHPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            out[u, v] = 1 / (1 + (rad / D) ** (2 * n))
    # retornar la matriz de salida
    return out

```

In [351...]:

```

ihpf_30 = IdealHighPassFilter(P, Q, 30)
ghpf_30 = GaussianHighPassFilter(P, Q, 30)
bhpf_30 = ButterworthHighPassFilter(P, Q, 30, 2.25)

```

In [352...]:

```

test_ihpf_30 = filtro_frecuencia(test, ihpf_30)
test_ghpf_30 = filtro_frecuencia(test, ghpf_30)
test_bhpf_30 = filtro_frecuencia(test, bhpf_30)

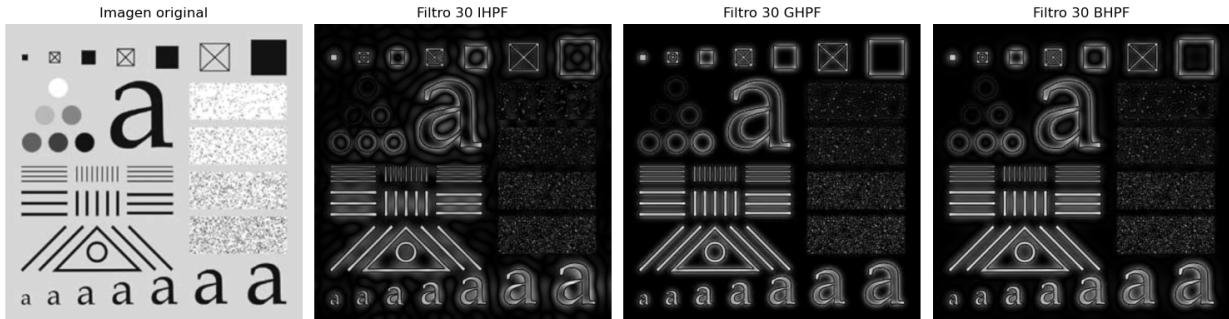
```

In [353...]:

```

show_images(test, test_ihpf_30, test_ghpf_30, test_bhpf_30,
            titles=['Imagen original', 'Filtro 30 IHPF', 'Filtro 30 GHPF',
                    'Filtro 30 BHPF'],
            cols=4, figsize=(15, 15))

```



4.9.2. El Laplaciano en el dominio de la frecuencia

En esta sección, revisamos el Laplaciano y mostramos que produce resultados equivalentes utilizando técnicas en el dominio de la frecuencia.

El laplaciano se puede implementar en el dominio de la frecuencia usando la función de transferencia de filtro.

$$H(u, v) = -4\pi^2 (u^2 + v^2)$$

o, con respecto al centro del rectángulo de frecuencia, usando la función de transferencia

$$\begin{aligned} H(u, v) &= -4\pi^2 [(u - P/2)^2 + (v - Q/2)^2] \\ &= -4\pi^2 D^2(u, v) \end{aligned}$$

donde $D(u, v)$ es la función de distancia definida previamente.

Usando esta función de transferencia, el Laplaciano de una imagen, $f(x, y)$, se obtiene de la manera familiar:

$$\nabla^2 f(x, y) = \mathcal{I}^{-1}[H(u, v)F(u, v)]$$

donde $F(u, v)$ es la DFT de $f(x, y)$.

Como anteriormente, la mejora se implementa usando la ecuación

$$g(x, y) = f(x, y) + c\nabla^2 f(x, y)$$

Aquí, $c = -1$ porque $H(u, v)$ es negativo.

Calcular $\nabla^2 f(x, y)$ con la ecuación. (4-125) introduce factores de escala DFT que pueden ser varios órdenes de magnitud mayores que el valor máximo de f .

- Por lo tanto, las diferencias entre f y su laplaciano deben llevarse a rangos comparables.
 - La forma más sencilla de manejar este problema es
 - normalizar los valores de $f(x, y)$ al rango $[0, 1]$ (antes de calcular su DFT) y
 - dividir $\nabla^2 f(x, y)$ por su valor máximo, lo que lo llevará al rango aproximado $[-1, 1]$
- .

Entonces se puede utilizar la ecuación anterior.

```
In [354]: def LaplacianHighPassFilter(P, Q):
    """
    Crear la funcion de transferencia Laplacian HPF.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
    Returns:
        np.ndarray: La funcion de transferencia LHPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
```

```

        out[u, v] = -4 * np.pi ** 2 * ((u - P / 2) ** 2 + (v - Q / 2) ** 2)
# retornar la matriz de salida
return out

```

In [355...]

```

def normalize(img, min_value=0, max_value=1):
    """
    Normalizar una imagen.
    Args:
        img (np.ndarray): La imagen de entrada.
        min_value (float): El valor mínimo de la imagen (default=0).
        max_value (float): El valor máximo de la imagen (default=1).
    Returns:
        np.ndarray: La imagen normalizada.
    """
    # obtener el valor mínimo y máximo de la imagen
    min_img = np.min(img)
    max_img = np.max(img)
    # normalizar la imagen
    out = (img - min_img) / (max_img - min_img) * (max_value - min_value) + min_value
    # retornar la imagen normalizada
    return out

```

In [356...]

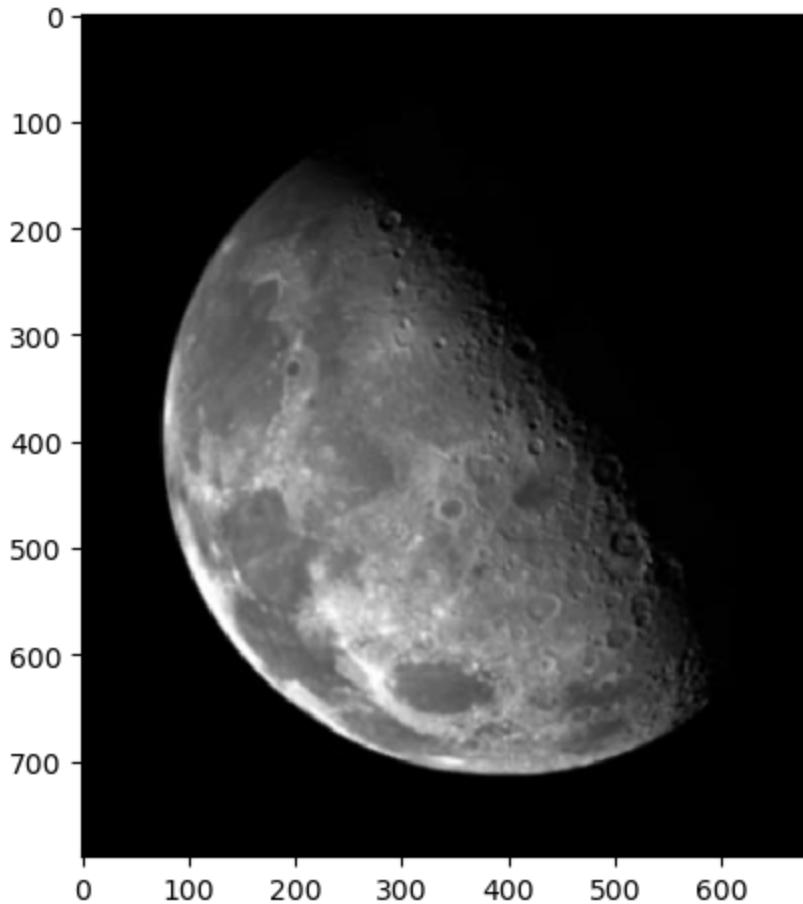
```

moon = imread('figs/moon.png', as_gray=True).astype(np.float32)
moon_norm = normalize(moon)
P = 2 * moon.shape[0]
Q = 2 * moon.shape[1]

imshow(moon_norm, cmap='gray')

```

Out[356]: <matplotlib.image.AxesImage at 0x140faf050>



```
In [357...]: lhpf = LaplacianHighPassFilter(P, Q)
moon_lhpf = filtro_frecuencia(moon_norm, lhpf)
moon_lhpf = moon_lhpf / np.max(moon_lhpf)
```

```
In [358...]: moon_ok = moon_norm + moon_lhpf
show_images(moon_norm, moon_ok,
            titles=['Imagen original', 'Imagen filtrada'],
            cols=2, figsize=(10, 10))
```



4.10. Filtrado selectivo

4.10.1. Filtros pasa banda y rechaza banda

Las funciones de transferencia de filtro de paso de banda y de rechazo de banda en el dominio de la frecuencia se pueden construir combinando funciones de transferencia de filtro de paso bajo y paso alto.

En otras palabras, las funciones de transferencia de filtro de paso bajo son la base para formar funciones de filtro de paso alto, rechazo de banda y paso de banda.

Además, una función de transferencia de filtro de paso de banda se obtiene a partir de una función de rechazo de banda de la misma manera que obtuvimos un paso alto a partir de una función de transferencia de paso bajo:

$$H_{\text{BP}}(u, v) = 1 - H_{\text{BR}}(u, v)$$

Filtro pasa banda ideal

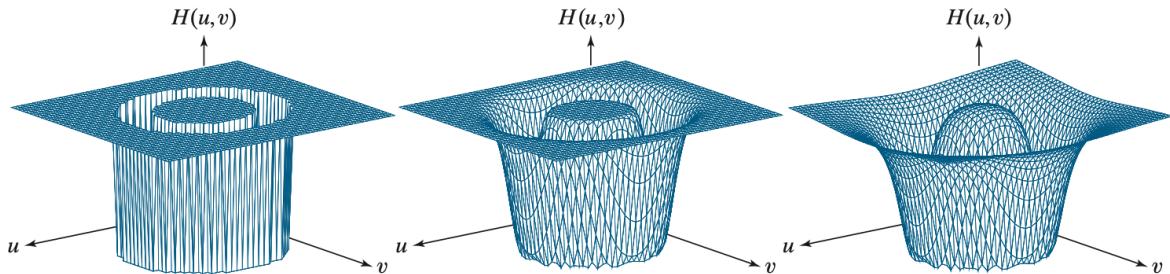
$$H(u, v) = \begin{cases} 0 & \text{if } C_0 - \frac{W}{2} \leq D(u, v) \leq C_0 + \frac{W}{2} \\ 1 & \text{dof} \end{cases}$$

Filtro pasa banda Gaussiano

$$H(u, v) = 1 - e^{-\left[\frac{D^2(u, v) - C_0^2}{D(u, v)W}\right]^2}$$

Filtro pasa banda Butterworth

$$H(u, v) = \frac{1}{1 + \left[\frac{D(u, v)W}{D^2(u, v) - C_0^2} \right]^{2n}}$$



```
In [359... def IdealBandRejectFilter(P, Q, c, W):
    """
    Crear la funcion de transferencia Ideal Band Reject Filter.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        c (int): El centro del filtro.
        W (int): El ancho del filtro.
    Returns:
        np.ndarray: La funcion de transferencia IBRF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            if D >= c - W / 2 and D <= c + W / 2:
                out[u, v] = 0
            else:
                out[u, v] = 1
    # retornar la matriz de salida
    return out
```

```
In [360... def GaussianBandRejectFilter(P, Q, c, W):
    """
    Crear la funcion de transferencia Gaussian Band Reject Filter.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        c (int): El centro del filtro.
        W (int): El ancho del filtro.
    Returns:
        np.ndarray: La funcion de transferencia GBRF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
```

```

        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            out[u, v] = 1 - np.exp(-((D**2 - c**2) / (D * W)) ** 2)
    # retornar la matriz de salida
    return out

```

In [361...]

```

def ButterworthBandRejectFilter(P, Q, c, W, n):
    """
    Crear la funcion de transferencia Butterworth Band Reject Filter.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        c (int): El centro del filtro.
        W (int): El ancho del filtro.
        n (int): El orden del filtro.
    Returns:
        np.ndarray: La funcion de transferencia BBRF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - P / 2) ** 2 + (v - Q / 2) ** 2)
            out[u, v] = 1 / (1 + ((D * W) / ((D ** 2) - (c ** 2))) ** (2 * n))
    # retornar la matriz de salida
    return out

```

In [362...]

```

def plot_filter(filter):
    filter = np.abs(filter)
    # create the x and y coordinate arrays (here we just use pixel indices)
    xx, yy = np.mgrid[0:filter.shape[0], 0:filter.shape[1]]

    # create the figure
    fig = plt.figure(figsize=(10, 10))
    ax = plt.axes(projection='3d')
    ax.plot_surface(xx, yy, filter, rstride=1, cstride=10, linewidth=0)

    # show it
    plt.show()

```

In [363...]

```

P = 2 * test.shape[0]
Q = 2 * test.shape[1]

c = 300
W = 200
n = 2

ibrf = IdealBandRejectFilter(P, Q, c, W)
gbrf = GaussianBandRejectFilter(P, Q, c, W)
bbrf = ButterworthBandRejectFilter(P, Q, c, W, n)

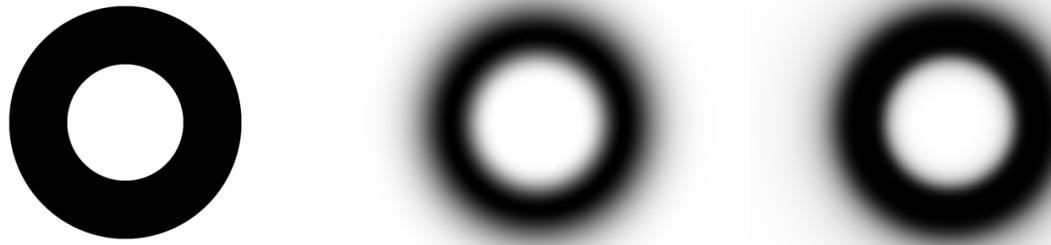
```

In [364...]

```

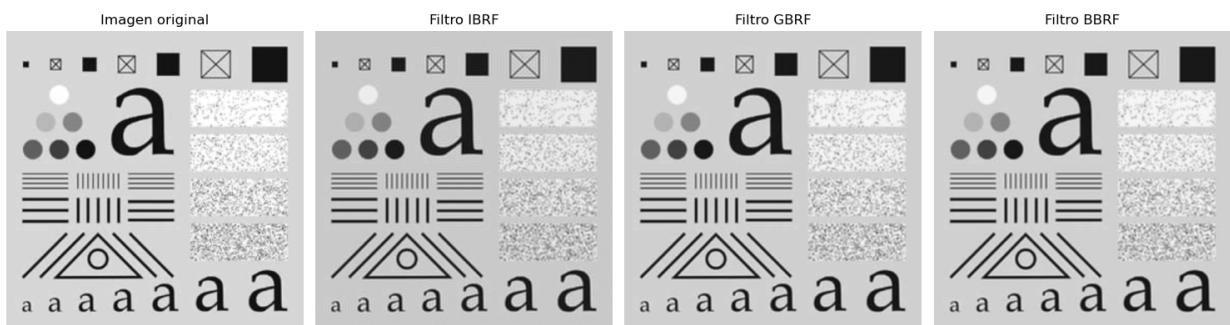
show_images(np.abs(ibrf), np.abs(gbrf), np.abs(bbrf),
           titles=['IBRF', 'GBRF', 'BBRF'],
           cols=3, figsize=(15, 15))

```



```
In [365...]: test_ibrf = filtro_frecuencia(test, ibrf)
test_gbrf = filtro_frecuencia(test, gbrf)
test_bbrf = filtro_frecuencia(test, bbrf)
```

```
In [366...]: show_images(test, test_ibrf, test_gbrf, test_bbrf,
                 titles=['Imagen original', 'Filtro IBRF', 'Filtro GBRF',
                         'Filtro BBRF'],
                 cols=4, figsize=(15, 15))
```



4.10.2. Filtros de muesca

Un filtro de muesca rechaza (o pasa) frecuencias en una vecindad predefinida del rectángulo de frecuencia.

Los filtros de desplazamiento de fase cero deben ser simétricos con respecto al origen (centro del rectángulo de frecuencia)

- por lo que una función de transferencia de filtro de muesca con centro en (u_0, v_0) debe tener una muesca correspondiente en la ubicación $(-u_0, -v_0)$.

Las funciones de transferencia del filtro de rechazo de muescas se construyen como productos de funciones de transferencia de filtros de paso alto cuyos centros se han trasladado a los centros de las muescas.

La forma general es:

$$H_{\text{NR}}(u, v) = \prod_{k=1}^Q H_k(u, v) H_{-k}(u, v)$$

donde $H_k(u, v)$ y $H_{-k}(u, v)$ son funciones de transferencia de filtro de paso alto cuyos centros están en (u_k, v_k) y $(-u_k, -v_k)$, respectivamente. Estos centros se especifican con respecto al centro del rectángulo de frecuencia, $(M/2, N/2)$, donde, como de costumbre, M y N son los números de filas y columnas en la imagen de entrada.

Por lo tanto, los cálculos de distancia para cada función de transferencia de filtro están dados por

$$D_k(u, v) = \left[(u - M/2 - u_k)^2 + (v - N/2 - v_k)^2 \right]^{1/2}$$

y

$$D_{-k}(u, v) = \left[(u - M/2 + u_k)^2 + (v - N/2 + v_k)^2 \right]^{1/2}$$

Por ejemplo, la siguiente es una función de transferencia de filtro de rechazo de muesca de Butterworth de orden n , que contiene tres pares de muescas:

$$H_{\text{NR}}(u, v) = \prod_{k=1}^3 \left[\frac{1}{1 + [D_{0k}/D_k(u, v)]^n} \right] \left[\frac{1}{1 + [D_{0k}/D_{-k}(u, v)]^n} \right]$$

La constante D_{0k} es la misma para cada par de muescas, pero puede ser diferente para diferentes pares.

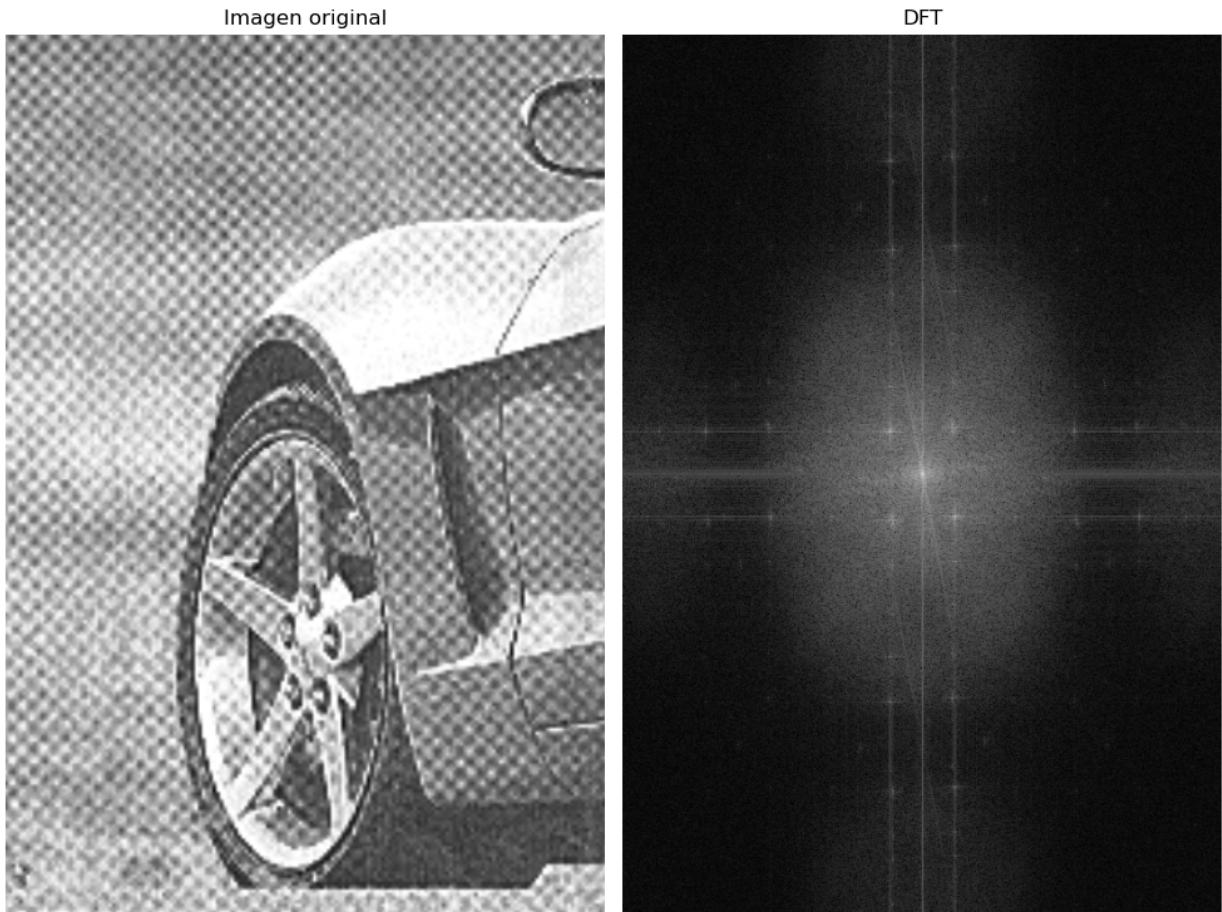
Una de las principales aplicaciones del filtrado de muescas es modificar selectivamente regiones locales de la DFT.

A menudo, este tipo de procesamiento se realiza de forma interactiva, trabajando directamente con DFT obtenidas sin relleno.

```
In [367...]: notch = imread('figs/notchImage.png', as_gray=True)
notch_dft = np.fft.fft2(shift(notch))

P = 2 * notch.shape[0]
Q = 2 * notch.shape[1]

show_images(notch, np.log(1 + np.abs(notch_dft)),
           titles=['Imagen original', 'DFT'],
           cols=2, figsize=(10, 10))
```



In [368...]

```
def ButterworthNotchFilter(P, Q, nu, nv, rad, n):
    """
    Crear la funcion de transferencia Butterworth HPF.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        rad (int): El radio del filtro.
        n (int): El orden del filtro.
    Returns:
        np.ndarray: La funcion de transferencia BHPF.
    """
    # inicializar la matriz de salida
    out = np.zeros((P, Q), dtype=np.complex64)
    # calcular la funcion de transferencia
    for u in range(P):
        for v in range(Q):
            D = np.sqrt((u - nu) ** 2 + (v - nv) ** 2)
            out[u, v] = 1 / (1 + (rad / D) ** (2 * n))
    # retornar la matriz de salida
    return out
```

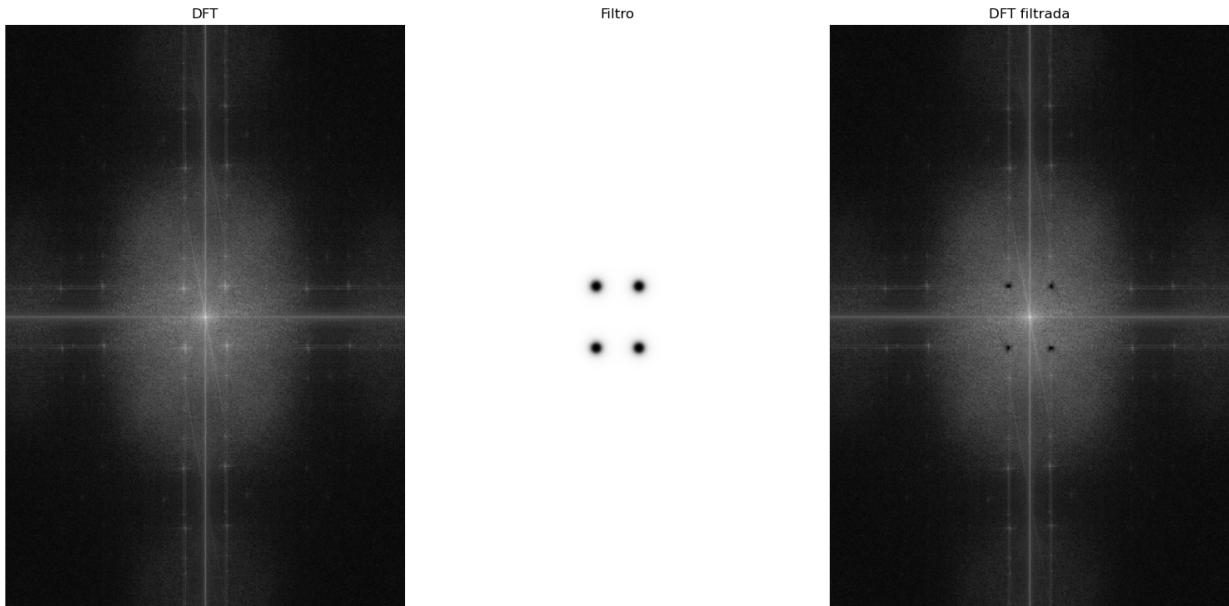
In [369...]

```
def NotchFilter(P, Q, notch):
    """
    Crear la funcion de transferencia Notch Filter.
    Args:
        P (int): El numero de filas de la imagen.
        Q (int): El numero de columnas de la imagen.
        notch (list): La lista de coordenadas de los notch.
    Returns:
        np.ndarray: La funcion de transferencia Notch Filter.
```

```
#####
# inicializar la matriz de salida
out = np.ones((P, Q), dtype=np.complex64)
# calcular la funcion de transferencia
for n in notch:
    u, v, r, nn = n
    out = out * ButterworthNotchFilter(P, Q, P / 2 + u, Q / 2 + v, r, nn)
    out = out * ButterworthNotchFilter(P, Q, P / 2 - u, Q / 2 - v, r, nn)
# retornar la matriz de salida
return out
```

In [370...]
`notches = [[84, 58, 15, 2], [-84, 58, 15, 2]]
notch_filter = NotchFilter(P, Q, notches)`

In [371...]
`notch_padded = np.pad(notch, ((0, P - notch.shape[0]), (0, Q - notch.shape[1])))
notch_dft = np.fft.fft2(shift(notch_padded))
notch_filtered = notch_dft * notch_filter
show_images(np.log(1 + np.abs(notch_dft)), np.abs(notch_filter), np.log(1 + np.abs(notch_filtered)),
titles=['DFT', 'Filtro', 'DFT filtrada'],
cols=3, figsize=(15, 15))`



In [372...]
`res = filtro_frecuencia(notch, notch_filter)`

In [373...]
`show_images(notch, res,
titles=['Imagen original', 'Imagen filtrada'],
cols=2, figsize=(10, 10))`

Imagen original



Imagen filtrada



Fin de la Unidad 4