Ejercicio 1

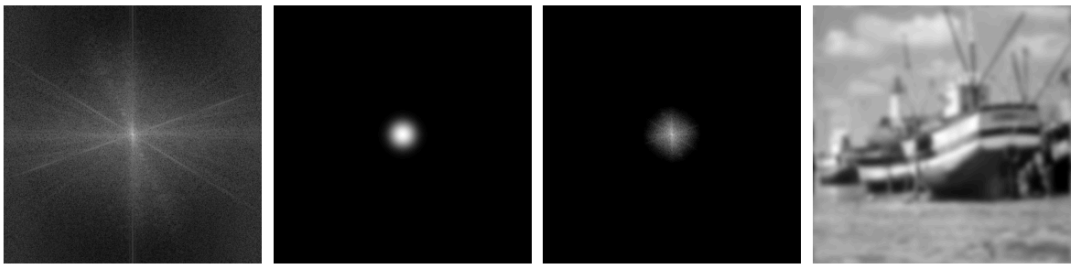
```
MASK_SIZE = 31
SIGMA = 5
```

```
# Convolución en el espacio
mascara = np.outer(scipy.signal.windows.gaussian(MASK_SIZE, SIGMA), scipy.signal.windows.gaussian(MASK_SIZE, SIGMA))
# Máscara de NxN con sigma 5
```

El código de este ejercicio es prácticamente igual al que tenemos en el ejemplo 3. El principal cambio se da en la creación de la variable sigma y el posterior uso de la misma en la creación de la convolución. Esta crea una matriz bidimensional que representa una máscara gaussiana.

Al aplicar np.outer a dos ventanas gaussianas generadas por scipy.signal.windows.gaussian, se obtiene una matriz bidimensional con el tamaño de la máscara y el ancho de la distribución gaussiana dados por las variables `MASK_SIZE` y `SIGMA`, que representa la máscara gaussiana.

Los resultados obtenidos son los siguientes:



Se muestran la imagen original, la máscara gaussiana en el dominio del espacio, el resultado de la convolución en el espacio, la magnitud de la Transformada de Fourier de la imagen original, la magnitud de la Transformada de Fourier de la máscara gaussiana y la magnitud de la Transformada de Fourier del resultado filtrado en el dominio de la frecuencia.

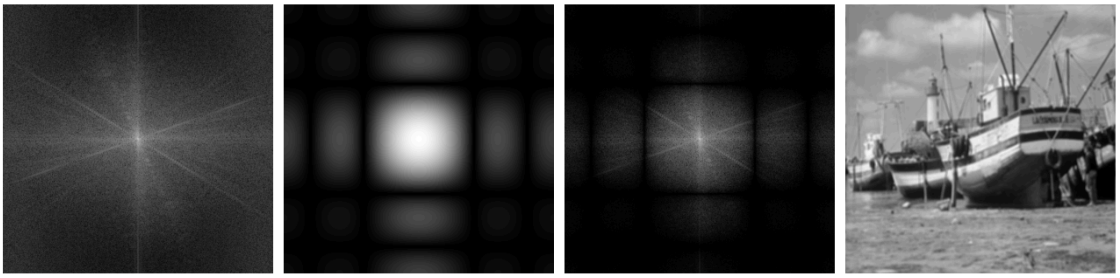
La máscara gaussiana suaviza una imagen aplicando un filtrado simétrico que se asemeja a una campana gaussiana, lo que resulta en una transición suave entre los píxeles. Por otro lado, la máscara media, al ser una operación de suavizado más agresiva, elimina secciones

de la imagen de manera más abrupta al promediar los valores de los píxeles en un área determinada.

Ejercicio 2

En este código el único cambio que existe con respecto al del anterior ejercicio es que se hace uso de una gaussiana de 5x5.

Los resultados obtenidos son los siguientes:



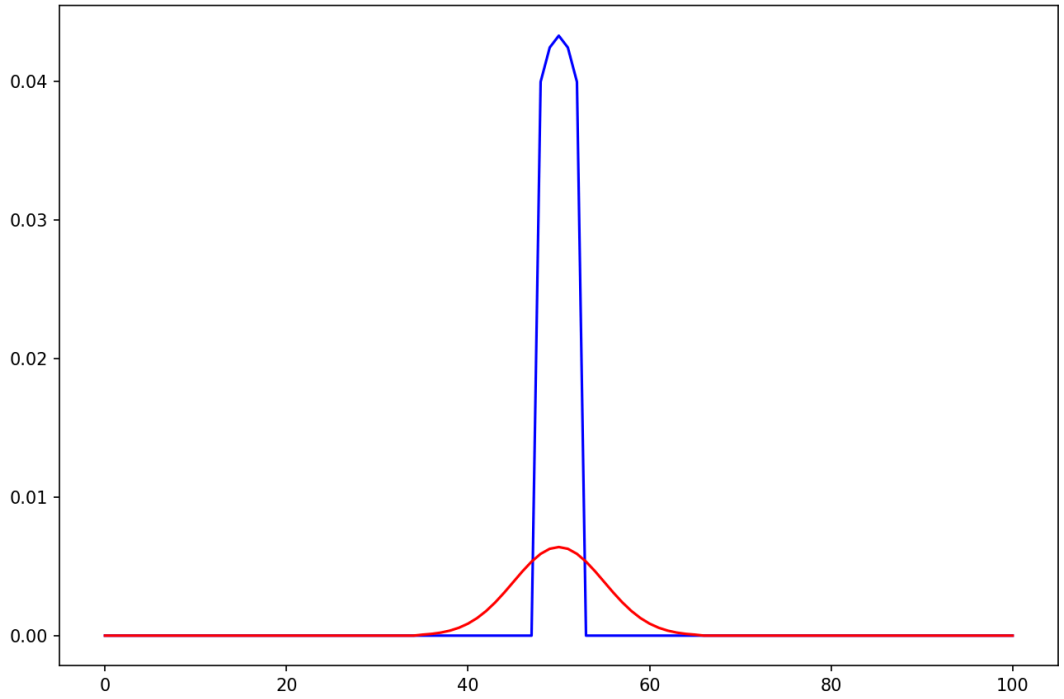
Al ser de un tamaño de máscara de 5, los efectos se asemejan más a los de la máscara media. Esto se debe a que la máscara gaussiana, al tener una dispersión pequeña, enfatiza más los píxeles cercanos, lo que resulta en una suavización que se asemeja a una eliminación abrupta de detalles, similar a lo que hace la máscara media.

Ejercicio 3

```python
def perfil_linea_central(size, MASK_SIZE, SIGMA):
    # Convolución en el espacio
    mascara = np.outer(scipy.signal.windows.gaussian(MASK_SIZE, SIGMA), scipy.signal.windows.gaussian(MASK_SIZE, SIGMA))
    mascara /= np.sum(mascara)  # Máscara normalizada

    # Ampliamos la máscara con ceros para que tenga el mismo tamaño que la imagen
    mascara_centrada = np.zeros((size, size))
    fila_i = size // 2 - MASK_SIZE // 2
    col_i = size // 2 - MASK_SIZE // 2
    mascara_centrada[fila_i:fila_i + MASK_SIZE, col_i:col_i + MASK_SIZE] = mascara

    perfil = mascara_centrada[size//2]
    return perfil


perfil_5 = perfil_linea_central(size, MASK_SIZES[0], SIGMA)
perfil_31 = perfil_linea_central(size, MASK_SIZES[1], SIGMA)
```

El código de este ejercicio viene siendo similar a los ya vistos anteriormente. La principal diferencia radica en que se crea una función capaz de calcular el perfil de la línea central. Primero, se define una máscara gaussiana bidimensional utilizando la función *scipy.signal.windows.gaussian* aplicada en dos dimensiones mediante *np.outer*. Esta máscara representa una distribución gaussiana en dos dimensiones. Luego, la máscara se normaliza, lo que garantiza que la suma de los elementos de la máscara sea igual a 1. Se crea una matriz de ceros del tamaño especificado. Se calculan los índices para colocar la máscara en el centro de la matriz de ceros y se coloca. A continuación, se extrae el perfil de la línea central de la máscara. Esto se hace seleccionando la fila en la mitad de la matriz de máscara (que representa la línea central). Y finalmente, la función devuelve el perfil de la línea central de la máscara.

Los resultados obtenidos son:



Algunos comentarios sobre las gráficas:

- El eje x representa la posición a lo largo del perfil de la línea central de la máscara.

- El eje y representa los valores de intensidad en el perfil.

Al observar las gráficas, se destacan las disparidades en las proporciones entre las dos máscaras gaussianas.

Se evidencia que la máscara más pequeña ejerce un impacto significativamente mayor en un área más limitada, lo que dificulta percibir claramente su forma como una función gaussiana. En otras palabras, al tratarse de un tamaño de máscara tan reducido como es un 5x5, aunque en el caso de un filtro gaussiano se trate generalmente de un círculo, se asemeja a un filtro de media pues es también una figura pequeña con forma de cuadrado.

Ejercicio 4

En este caso, tampoco existen cambios muy grandes en el código.

```
# Definir los tamaños de las máscaras
mask_sizes = list(range(3, 22, 2))

imagen = ski.io.imread("images/boat.511.tiff")
imagen = ski.util.img_as_float(imagen)

# Listas para almacenar los tiempos de ejecución
tiempos_espacio = []
tiempos_frecuencia = []

# Repetir cada convolución 10 veces para obtener un promedio
repeticiones = 10
```

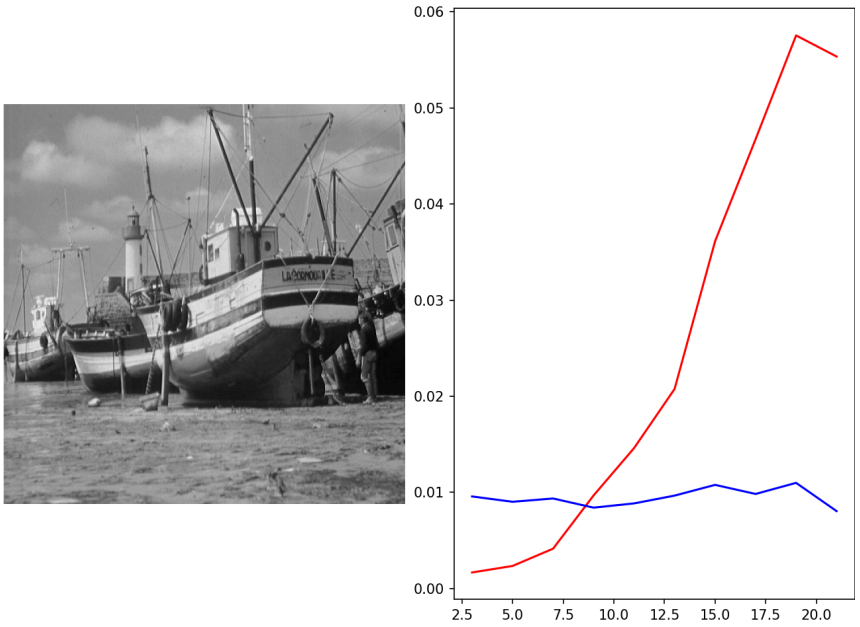
Aquí se definen los tamaños a usar en las máscaras, se crean unas listas para almacenar los tiempos de ejecución y se crea una variable repeticiones que se usará para obtener un promedio.

```
for mask_size in mask_sizes:
    # Convolución en el espacio
    mascara = np.ones((mask_size, mask_size))  # Máscara de NxN toda con 1
    mascara /= np.sum(mascara)  # Máscara normalizada

    # Medir el tiempo de ejecución para la convolución en el espacio
    inicio_espacio = time.time()
    for x in range(repeticiones):
        res_convol = scipy.ndimage.convolve(imagen, mascara, mode="wrap")
    fin_espacio = time.time()

    tiempo_promedio_espacio = (fin_espacio - inicio_espacio) / repeticiones
    tiempos_espacio.append(tiempo_promedio_espacio)

    # Ampliamos la máscara con ceros para que tenga el mismo tamaño que la imagen
    mascara_centrada = np.zeros(imagen.shape)
    fila_i = imagen.shape[0] // 2 - mask_size // 2
    col_i = imagen.shape[1] // 2 - mask_size // 2
    mascara_centrada[fila_i:fila_i + mask_size, col_i:col_i + mask_size] = mascara

    # Pasamos imagen y máscara a las frecuencias
    FTimagen = fft.fft2(imagen)
    mascara_en_origen = fft.ifftshift(mascara_centrada)
    FTmascara = fft.fft2(mascara_en_origen)

    # Medir el tiempo de ejecución para la convolución en las frecuencias
    inicio_frecuencia = time.time()
    for x in range(repeticiones):
        FTimagen_filtrada = FTimagen * FTmascara  # Producto punto a punto
        res_filtro_FT = fft.ifft2(FTimagen_filtrada)
    fin_frecuencia = time.time()

    tiempo_promedio_frecuencia = (fin_frecuencia - inicio_frecuencia) / repeticiones
    tiempos_frecuencia.append(tiempo_promedio_frecuencia)
```

Este código realiza una comparación entre los tiempos de ejecución de dos métodos de filtrado de imágenes: convolución en el espacio y convolución en el dominio de la frecuencia.

Para esto se crea una máscara de tamaño NxN compuesta completamente de unos (np.ones((mask_size, mask_size))) y se normaliza dividiendo por la suma de todos los elementos. Luego se mide el tiempo de ejecución promedio para realizar la convolución en el espacio usando scipy.ndimage.convolve(). Esto se hace varias veces (repeticiones) para obtener un promedio más preciso. Se amplía la máscara con ceros para que tenga el mismo tamaño que la imagen. Se realiza la transformada de Fourier 2D tanto de la imagen original como de la máscara, y se desplaza la máscara al origen de la frecuencia. Por último, se mide el tiempo de ejecución promedio para realizar la convolución en el dominio de la frecuencia y del espacio.

Los resultados son los siguientes:



Comentarios sobre la Diferencia de Tiempos de Ejecución:

- Se puede observar que para tamaños de máscara pequeños, las convoluciones en el espacio y en frecuencia tienen tiempos de ejecución similares.
- Sin embargo, a medida que aumenta el tamaño de la máscara, la convolución en el espacio se vuelve significativamente más lenta en comparación con la convolución en el dominio de la frecuencia.
- Esto demuestra la ventaja de utilizar la convolución en el dominio de la frecuencia para tamaños de máscara más grandes, ya que es computacionalmente más eficiente en estos casos.

Ejercicio 5

```
# Filtro paso alto
mascara_centrada_alto = 1 - crear_circulo_centrado(imagen.shape, FREQ_ALTA_EN_PIXELES)
FT_imagen_filtradaAlto = FTimagen_centrada * mascara_centrada_alto
imagen_filtrada_alto = fft.ifft2(fft.ifftshift(FT_imagen_filtradaAlto))
imagen_filtrada_alto_real = np.real(imagen_filtrada_alto)
imagen_filtrada_alto_imag = np.imag(imagen_filtrada_alto)
if not np.allclose(imagen_filtrada_alto_imag, np.zeros(imagen.shape)):
    print("Warning. Algo no está yendo bien!!")

# Filtro de paso banda
mascara_centrada_banda = (crear_circulo_centrado(imagen.shape, FREQ_ALTA_EN_PIXELES) -
                          crear_circulo_centrado(imagen.shape, FREQ_CORTE_EN_PIXELES))
FT_imagen_filtradaBanda = FTimagen_centrada * mascara_centrada_banda
imagen_filtrada_banda = fft.ifft2(fft.ifftshift(FT_imagen_filtradaBanda))
imagen_filtrada_banda_real = np.real(imagen_filtrada_banda)
imagen_filtrada_banda_imag = np.imag(imagen_filtrada_banda)
if not np.allclose(imagen_filtrada_banda_imag, np.zeros(imagen.shape)):
    print("Warning. Algo no está yendo bien!!")

# Suma de los resultados de los tres filtros
suma = imagen_filtrada_bajo + imagen_filtrada_alto + imagen_filtrada_banda
```

Este código sí que varía un poco más con respecto a los anteriores. Se crean dos filtros.

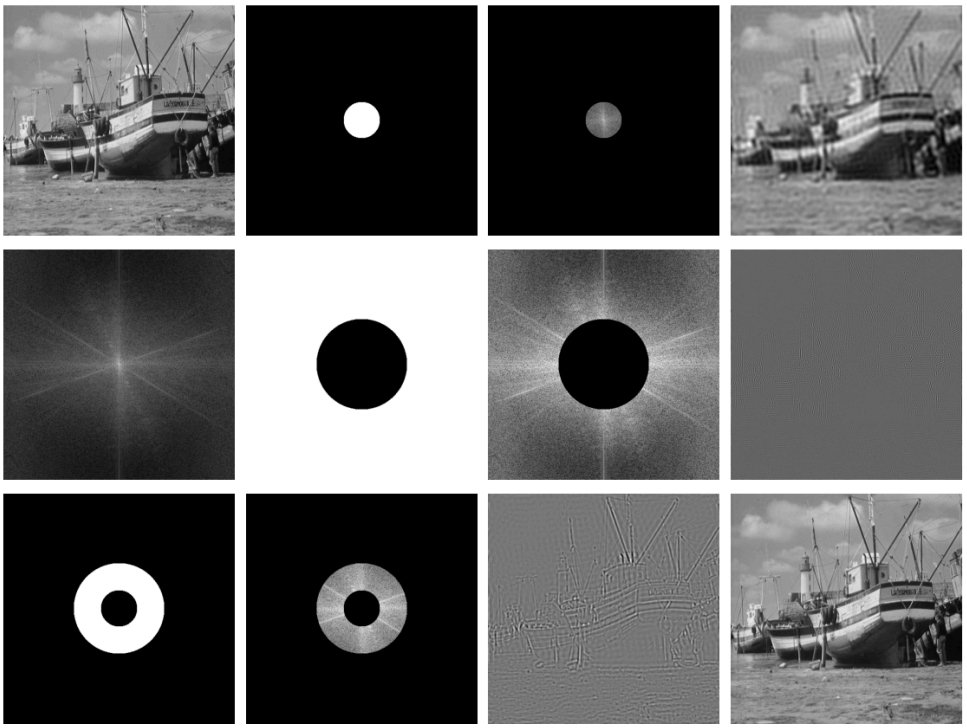
- Filtro Paso Alto:

Se crea una máscara que representa un filtro paso alto. Esta máscara se obtiene restando un círculo centrado en frecuencias bajas (que representa las frecuencias que queremos atenuar) de una matriz de unos. Esto permite que solo las frecuencias altas pasen a través del filtro. Luego, se multiplica la transformada de Fourier de la imagen original por esta máscara en el dominio de la frecuencia y se aplica la transformada inversa de Fourier al resultado para obtener la imagen filtrada en el dominio espacial.

- Filtro de Paso Banda:

Se crea una máscara que representa un filtro de paso banda. Esta máscara se obtiene restando un círculo centrado en frecuencias bajas y otro círculo centrado en frecuencias altas (definido por una banda de frecuencia) de una matriz de unos. Esto permite que solo las frecuencias dentro de la banda deseada pasen a través del filtro. Luego, se multiplica la transformada de Fourier de la imagen original por esta máscara en el dominio de la frecuencia y se aplica la transformada inversa de Fourier al resultado para obtener la imagen filtrada en el dominio espacial.

Los resultados obtenidos son los siguientes:



El resultado obtenido se debe a la propiedad de linealidad de la transformada de Fourier.

La propiedad de linealidad de la transformada de Fourier establece que la transformada de Fourier de una combinación lineal de señales es igual a la combinación lineal de las transformadas de Fourier de esas señales individuales. En otras palabras, si tienes dos señales f(x) y g(x), y una constante c, entonces:

$$\mathcal{F}(c \cdot f(x) + g(x)) = c \cdot \mathcal{F}(f(x)) + \mathcal{F}(g(x))$$

Dicho de otra forma, la imagen filtrada mediante la máscara de suma muestra una similitud exacta con la imagen original y esto se explica por el hecho de que al sumar las frecuencias contenidas en estas regiones adyacentes, se obtiene una representación que es idéntica a la imagen original.

Estudiante: Irene Mejía Hurtado
Profesor: Pedro García Sevilla