ex1-2

October 15, 2024

1 ex1: Inverse Filtering

[1]:
```
from scipy.fft import fft2, fftshift
from pathlib import Path
import numpy as np
import skimage
import matplotlib.pyplot as plt
```

[2]:
```
ASSETS_FOLDER_PATH = "./assets"
OUTPUT_FOLDER_PATH = "."
```

[3]:
```
Path(OUTPUT_FOLDER_PATH).mkdir(parents=True, exist_ok=True)
```

Cargamos la imagen

[4]:
```
my_image = skimage.io.imread(fname=f"{ASSETS_FOLDER_PATH}/lena_gray.tif")
imsize = my_image.shape
```

Definimos las funciones a utilizar:

[5]:
```
def fft2c(x: np.ndarray) -> np.ndarray:
    return fftshift(fft2(fftshift(x)))
```

[6]:
```
def ifft2c(x: np.ndarray) -> np.ndarray:
    return fftshift(np.fft.ifft2(fftshift(x)))
```

[7]:
```
def lowpass_kernel(kernel_size, image_dimensions):
    height, width = image_dimensions
    kernel = np.zeros((height, width))
    center_x, center_y = (height // 2, width // 2)
    kernel[center_x - kernel_size // 2: center_x + kernel_size // 2 + 1,
           center_y - kernel_size // 2: center_y + kernel_size // 2 + 1] = 1 /
 ↪(kernel_size ** 2)
    return kernel
```

[8]:
```
def plot_image(img, title):
    plt.imshow(img, cmap='gray')
    plt.axis('off')
```

```
    plt.title(title)
    plt.show()
```

[9]:
```
def plot_images(img1, title1, img2, title2):
    fig = plt.figure(figsize=(6, 7))

    fig.add_subplot(1, 2, 1)
    plt.imshow(img1, cmap='gray')
    plt.axis('off')
    plt.title(title1)

    fig.add_subplot(1, 2, 2)
    plt.imshow(img2, cmap='gray')
    plt.axis('off')
    plt.title(title2)

    plt.subplots_adjust(wspace=0.05, hspace=0)
    plt.show()
```

[10]:
```
def mse(original, variation):
    return np.mean((original - variation) ** 2)
```

1.1 Blur

Calculamos el kernel:

[11]:
```
gks = 80
gk = np.exp(-(np.arange(max(imsize)) - max(imsize) // 2) ** 2 / (2 * gks ** 2))
gauss_kernel = np.outer(gk, gk)

gauss_kernel = gauss_kernel[
    (max(imsize) - imsize[0]) // 2:(max(imsize) - imsize[0]) // 2 + imsize[0],
    (max(imsize) - imsize[1]) // 2:(max(imsize) - imsize[1]) // 2 + imsize[1]
]
```

[12]:
```
# blur_kernel = gauss_kernel
blur_kernel = lowpass_kernel(5, imsize)
```

Obtenemos la imagen con blur:

[13]:
```
blurred_image = np.abs(ifft2c(fft2c(my_image) * fft2c(blur_kernel)))
plot_images(my_image, 'Original image', blurred_image, 'Blurred image')
```

Aplicando la operación inversa sobre la imagen borrosa:

[14]:
```
restored_image_blur = np.abs(ifft2c(fft2c(blurred_image) / fft2c(blur_kernel)))
plot_images(blurred_image, 'Blurred image', restored_image_blur, 'Restored image')
print(f"mse: {mse(my_image, restored_image_blur)}")
```


```
mse: 1.2837269174207216e-21
```

1.2 Blur + Noise

Definimos el signal-to-noise ratio:

```
[15]: SNR_dB = 10
```

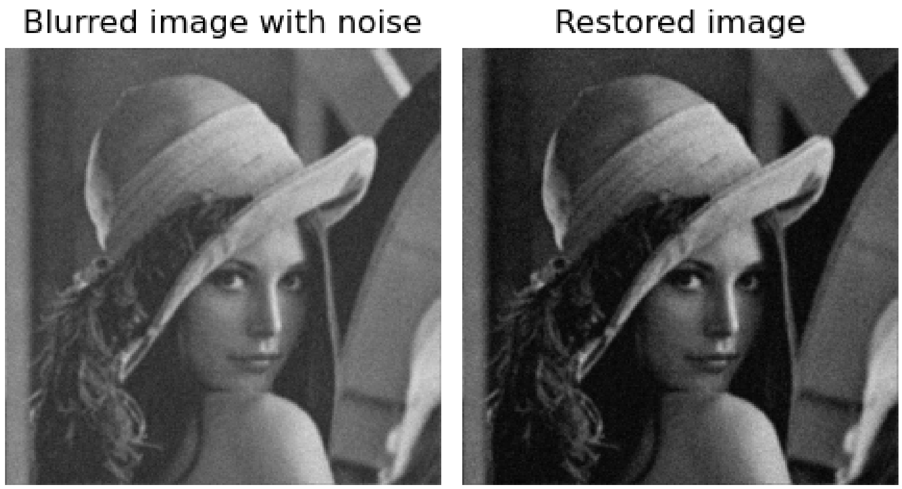
Calculamos la desviación estándar de la imagen borrosa para el ruido:

```
[16]: sigma_blurred_image = np.std(blurred_image)
      sigma_noise = np.sqrt((sigma_blurred_image ** 2) * (10 ** (-SNR_dB / 10)))
      noise = np.random.normal(0, sigma_noise, (imsize[0], imsize[1]))
```

```
[17]: blurred_noise_image = blurred_image + noise
      plot_images(my_image, 'Original image', blurred_noise_image, 'Blurred image␣
       ↪with noise')
```

Aplicando la operación inversa:

```
[18]: restored_image_blur_noise = np.abs(ifft2c(fft2c(blurred_noise_image)))
      plot_images(blurred_noise_image, 'Blurred image with noise',␣
       ↪restored_image_blur_noise, 'Restored image')
      print(f"mse: {mse(my_image, restored_image_blur_noise)}")
```



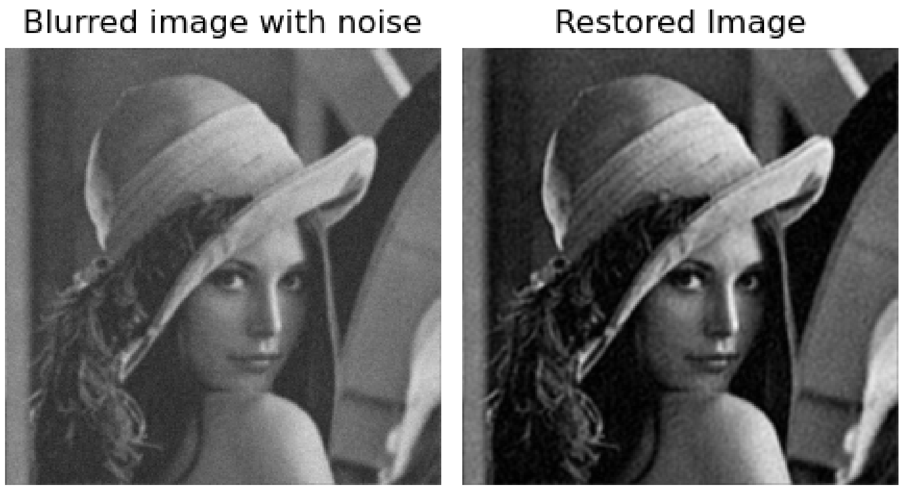

```
mse: 273.99873726572713
```

2 ex2: Wiener

Definimos un filtro de Wiener:

[19]:
```
def wiener_filter_restoration(blurred_image, blur_kernel, noise):
    blurred_fft = fft2c(blurred_image)
    kernel_fft = fft2c(blur_kernel)
    noise_psd = np.square(np.abs(fft2c(noise)))
    wiener_filter = np.conj(kernel_fft) / (np.abs(kernel_fft) ** 2 + noise_psd /
 ↪ np.square(np.abs(blurred_fft)))
    restored_fft = wiener_filter * blurred_fft
    restored_image = np.abs(ifft2c(restored_fft))
    return restored_image
```

[20]:
```
restored_image_wiener = wiener_filter_restoration(blurred_noise_image,␣
 ↪blur_kernel, noise)
plot_images(blurred_noise_image, 'Blurred image with noise',␣
 ↪restored_image_wiener, 'Restored Image')
print(f"mse: {mse(my_image, restored_image_wiener)}")
```




```
mse: 68.87812877972368
```