

1a Lista de Exercícios de Processamento Digital de Imagens – 2023-1 - Graduação

Lara De Luca Mota Teixeira

Data de Entrega: 04/06/2022

Preparação para as questões:

- Montando a pasta do drive que será usada para acessar as imagens usadas
- Importando as bibliotecas usadas

In [2]:

```
# Montando o drive
from google.colab import drive
drive.mount('/content/drive')

# Caminho da pasta onde as imagens usadas estão
list_path = '/content/drive/MyDrive/PDI/Lista 1/'

# Imports
from matplotlib import pyplot as plt
import numpy as np
```

Mounted at /content/drive

1)

Compare os três métodos clássicos de interpolação: *vizinho mais próximo, bilinear e bicúbica*.

Reduza o tamanho da imagem Fig10.29(a).jpg em 4 vezes usando o método vizinho mais próximo, em seguida, aumente o tamanho da imagem resultante em 4 vezes utilizando o método vizinho mais próximo (para ela voltar ao tamanho original). Repita o procedimento para os outros dois métodos de interpolação. Pode usar função pronta para isto. Exiba as imagens obtidas e a imagem original.

Compare os resultados obtidos por cada método e com a imagem original.

Qual foi o método que você achou melhor e por quê?

Qual foi o pior e por quê?

Resposta:

In []:

```
from skimage.transform import rescale
image = plt.imread(list_path+'Fig10.29(a).jpg')
#0: Nearest-neighbor
```

```

#1: Bi-Linear (default)
#2: Bi-quadratic
#3: Bi-cubic
#4: Bi-quartic
#5: Bi-quintic

# Vizinho mais proximo
im_down_nn = rescale(image, scale=0.25, anti_aliasing=True, order=0)
im_up_nn = rescale(im_down_nn, scale=4, anti_aliasing=True, order=0)

# Bilinear
im_down_bilin = rescale(image, scale=0.25, anti_aliasing=True, order=1)
im_up_bilin = 255*rescale(im_down_bilin, scale=4, anti_aliasing=True, order=1)

# Bicubica
im_down_biCub = rescale(image, scale=0.25, anti_aliasing=True, order=3)
im_up_biCub = 255*rescale(im_down_biCub, scale=4, anti_aliasing=True, order=3)

fig, ax = plt.subplots(3, 2, figsize=(8,12))
ax[0, 0].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[0, 0].set_title('Imagen Original')
ax[0, 0].axis('off')
ax[0, 1].imshow(im_up_nn, cmap='gray', vmin=0, vmax=255)
ax[0, 1].set_title('Vizinho mais próximo')
ax[0, 1].axis('off')

ax[1, 0].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[1, 0].set_title('Imagen Original')
ax[1, 0].axis('off')
ax[1, 1].imshow(im_up_bilin, cmap='gray', vmin=0, vmax=255)
ax[1, 1].set_title('Bilinear')
ax[1, 1].axis('off')

ax[2, 0].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[2, 0].set_title('Imagen Original')
ax[2, 0].axis('off')
ax[2, 1].imshow(im_up_biCub, cmap='gray', vmin=0, vmax=255)
ax[2, 1].set_title('Bicúbica')
ax[2, 1].axis('off')

```

Out[]: (-0.5, 399.5, 479.5, -0.5)

Imagen Original



Vizinho mais próximo



Imagen Original



Bilinear



Imagen Original



Bicúbica



Qual foi o método que você achou melhor e por quê?

- O melhor método foi a interpolação bicúbica, pois é o que apresenta maior semelhança a imagem original. O método de vizinho mais próximo obtém uma imagem com muita distorção nas bordas da digital na imagem, e o método bilinear obtém uma imagem com cores menos próximas a original, devido a menor nitidez em relação ao método de interpolação bicúbica.

Qual foi o pior e por quê?

- Já o pior foi o método de vizinho mais próximo, pois obtém uma imagem com muita distorção em relação aos outros métodos.
-

2)

Use a técnica de fatiamento de níveis de intensidade para realçar a aorta da imagem Fig10.15(a).jpg.

Resposta:

```
In [ ]: ##### Fatiamento de Níveis de Intensidade #####
# Fatiamento binario
def fatiamentoBin(image, low, high):
    resul = image.copy()
    for i in range(resul.shape[0]):
        for j in range(resul.shape[1]):
            if resul[i, j] < low or resul[i, j] > high:
                resul[i, j] = 0
            else: # image[i][j] >= Low and image[i][j] <= high
                resul[i, j] = 255
    return resul

# Fatiamento Linear
def fatiamentoLinear(image, low, high):
    resul = image.copy()
    for i in range(resul.shape[0]):
        for j in range(resul.shape[1]):
            if resul[i, j] >= low and resul[i, j] <= high:
                resul[i, j] = 255
    return resul
```

```
In [ ]: # Lendo as imagens:
image = plt.imread(list_path+'Fig10.15(a).jpg')
image = image.astype(np.float32) # ou np.float64

# Processando a imagem:
low = 148 # threshold minimo
high = 200 # threshold maximo
result1 = fatiamentoBin(image, low, high)

# range_thr = 16
# low = np.median(image)-range_thr # threshold minimo
# high = np.median(image)+range_thr # threshold maximo
result2 = fatiamentoLinear(image, low, high)

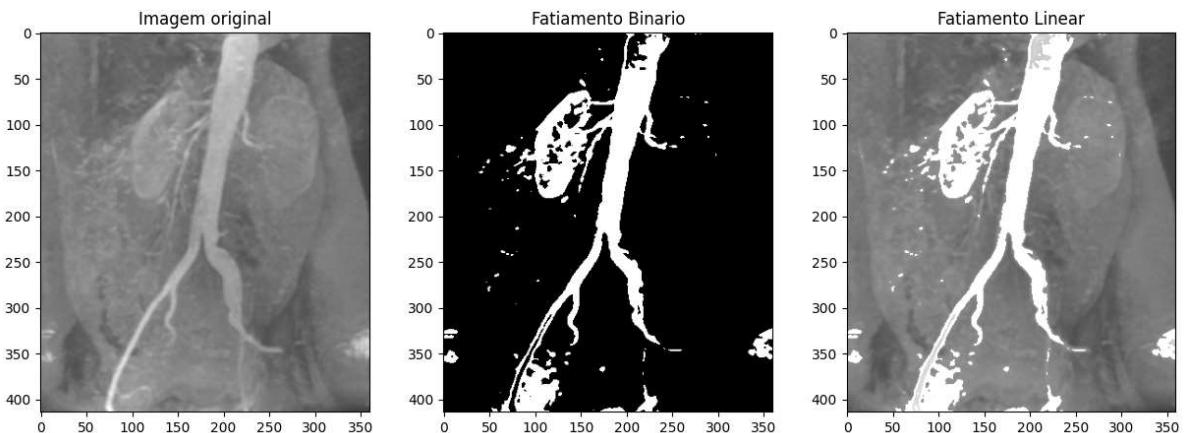
# Exibir imagem
```

```

fig, ax = plt.subplots(1, 3, figsize=(15,5))
ax[0].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[0].set_title('Imagen original')
ax[1].imshow(resul1, cmap='gray', vmin=0, vmax=255)
ax[1].set_title('Fatiamento Binario')
ax[2].imshow(resul2, cmap='gray', vmin=0, vmax=255)
ax[2].set_title('Fatiamento Linear')

```

Out[]: Text(0.5, 1.0, 'Fatiamento Linear')



3)

Faça uma rotina que implemente uma máscara de convolução espacial de dimensão NxN (N ímpar).

Discute possíveis soluções para tratamento de bordas e implemente uma delas.

Teste a rotina implementada na imagem lena.tif, para os seguintes casos:

- a) filtro passa-baixas,
- b) filtro laplaciano (adicionando a imagem original ao resultado),
- c) filtro de Sobel (passe as duas máscaras e retorne a imagem da magnitude do gradiente).

Exiba os resultados obtidos e analise-os.

Resposta:

O tratamento de borda envolve preencher a imagem com uma camada de pixels envolta das bordas para que ao realizar a convolução espacial o resultado mantenha as mesmas dimensões da imagem original. Algumas abordagens são:

- Padding com zeros: Preencher a camada extra com zeros;
 - Imagem original

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- 1 camada de padding de zeros

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & \textcolor{red}{1} & \textcolor{red}{2} & \textcolor{red}{3} & 0 \\ 0 & \textcolor{red}{4} & \textcolor{red}{5} & \textcolor{red}{6} & 0 \\ 0 & \textcolor{red}{7} & \textcolor{red}{8} & \textcolor{red}{9} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Padding reflexivo: Preencher a camada extra com o número do lado contrário ao do padding, por exemplo:

- Abaixo, o primeiro elemento da matriz é '1', à sua direita o elemento é '2', assim, o valor preenchido à esquerda do primeiro elemento será '2'.
- Já para o elemento (2, 3) = '6', à sua direita, o valor a ser preenchido será o valor à sua esquerda, ou seja, '5'.
- Por fim, para o elemento (1, 3) = '3', na sua diagonal superior direita, o elemento que será preenchido será o de sua diagonal inferior esquerda, ou seja, '5'.
- Imagem original

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- 1 camada de padding reflexivo

$$\begin{bmatrix} 5 & 4 & 5 & 6 & 5 \\ 2 & \textcolor{red}{1} & \textcolor{red}{2} & \textcolor{red}{3} & 2 \\ 5 & \textcolor{red}{4} & \textcolor{red}{5} & \textcolor{red}{6} & 5 \\ 8 & \textcolor{red}{7} & \textcolor{red}{8} & \textcolor{red}{9} & 8 \\ 5 & 4 & 5 & 6 & 5 \end{bmatrix}$$

- Padding de wrap: Preencher a camada extra com valores 'circulares', ou seja, o padding ao lado de um número terá o valor do último elemento da matriz naquela mesma direção. Alguns exemplos:

- Abaixo, o primeiro elemento da matriz é '1', o elemento a ser preenchido acima dele, ou seja, na mesma **coluna**, será o último elemento daquela coluna = '7';
- Já para o elemento (2, 3) = '6', à sua direita, ou seja, na mesma **linha**, será o primeiro valor da linha em que o elemento está = '4'.
- Por fim, para o elemento (1, 3) = '3', na sua diagonal superior direita, ou seja na mesma **diagonal**, será o último elemento daquela diagonal = '7'.
- Imagem original

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- 1 camada de padding:

$$\begin{bmatrix} 9 & 7 & 8 & 9 & 7 \\ 3 & \textcolor{red}{1} & \textcolor{red}{2} & \textcolor{red}{3} & 1 \\ 6 & \textcolor{red}{4} & \textcolor{red}{5} & \textcolor{red}{6} & 4 \\ 9 & \textcolor{red}{7} & \textcolor{red}{8} & \textcolor{red}{9} & 7 \\ 3 & 1 & 2 & 3 & 1 \end{bmatrix}$$

- 2 camadas de padding:

$$\begin{bmatrix} 5 & 6 & 4 & 5 & 6 & 4 & 5 \\ 8 & 9 & 7 & 8 & 9 & 7 & 8 \\ 2 & 3 & \textcolor{red}{1} & \textcolor{red}{2} & \textcolor{red}{3} & 1 & 2 \\ 5 & 6 & \textcolor{red}{4} & \textcolor{red}{5} & \textcolor{red}{6} & 4 & 5 \\ 8 & 9 & \textcolor{red}{7} & \textcolor{red}{8} & \textcolor{red}{9} & 7 & 8 \\ 2 & 3 & 1 & 2 & 3 & 1 & 2 \\ 5 & 6 & 4 & 5 & 6 & 4 & 5 \end{bmatrix}$$

```
In [31]: #####
# Mascara de convolucao espacial
...
Aplica uma mascara convolucional a uma imagem
* inputs:
  * imagem
  * kernel
* output: imagem de entrada modificada
* pre-condicao: imagem valida (diferente de None) e mascara valida (resolucao
  NxN com N impar)
* pos-condicao: imagem de saida foi alterada de acordo com a mascara
...
def convEspacial(image, kernel):
    k_lin, k_col = kernel.shape
    if k_lin != k_col:
        print("Mascara invalida, a funcao so suporta mascaras de mesma dimensao em x e
              y")
    # VALIDADAS AS CONDICOES, INICIA-SE A OPERACAO DE FATO
    # Cria uma copia da imagem original para alocar a imagem resultante
    lin, col = image.shape
    resultado = np.zeros((lin, col))

    # Fazendo o padding de zeros na imagem
    padding_size = int((k_col-1)/2)
    image_padded = np.zeros((lin+2*padding_size, col+2*padding_size))
    image_padded[padding_size:lin+padding_size, padding_size:col+padding_size] = image

    # Percorre a imagem com padding aplicando a transformacao
    lin, col = image_padded.shape

    for i in range(lin-2*padding_size):
        for j in range(col-2*padding_size):
            sub_area = image_padded[i:i+k_lin, j:j+k_col]
            resultado[i, j] = np.sum(sub_area*kernel)

    return resultado
#####
# Filtro Passa-Baixas (ou Filtro de Media)
...
Aplica um filtro do tipo Passa-Baixas (ou de Media) em uma imagem
* inputs:
  * imagem
  * resolucao da mascara a ser aplicada
```

```

    * output: imagem de entrada modificada
    * pre-condicao: imagem valida (diferente de None) e resolucao valida (impar)
    * pos-condicao: imagem de saida foi alterada de acordo com a mascara
    ...

def filtro_PassaBaixas(image, res = 3):
    if res%2 != 1:
        print("Resolucao invalida! \nA resolucao do filtro deve ser um valor impar!")
        return -1

    # Filtro de media, ou Filtro Passa-Baixas
    kernel = np.ones((res, res))
    kernel = kernel/(res**2)

    resul = convEspacial(image, kernel)
    return resul

#####
# Filtro Laplaciano
...
Aplica um filtro do tipo Laplaciano em uma imagem
    * inputs: imagem
    * output: imagem de entrada modificada
    * pre-condicao: imagem valida (diferente de None)
    * pos-condicao: imagem de saida foi alterada de acordo com a mascara
    ...

def filtro_Laplaciano(image):
    # Filtro Laplaciano com adicao da imagem original
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])

    resul = convEspacial(image, kernel)
    return resul

#####
# Filtro de Sobel (ou Filtro Gradiente)
...
Aplica um Filtro de Sobel (ou Gradiente), em ambas as direcoes, em uma imagem
    * inputs: imagem
    * output: imagem de entrada modificada
    * pre-condicao: imagem valida (diferente de None)
    * pos-condicao: imagem de saida foi alterada de acordo com a mascara
    ...

def filtro_Sobel(image):
    # Filtro de Sobel, ou Filtro de Gradiente (ambas direcoes)
    kernel_x = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    kernel_y = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

    resul_x = convEspacial(image, kernel_x)
    resul_y = convEspacial(image, kernel_y)

    resul = np.sqrt(resul_x**2 + resul_y**2)

    return resul

```

```

In [ ]: # Lendo as imagens:
image = plt.imread(list_path+'lena.tif')
image = image.astype(np.float32) # ou np.float64

# Processando a imagem:
res = 3 # tamanho do filtro, deve ser um valor impar
result1 = filtro_PassaBaixas(image, res = res)
result2 = filtro_Laplaciano(image)
result3 = filtro_Sobel(image)

```

```
In [ ]: # Exibir imagem
fig, ax = plt.subplots(2, 2, figsize=(10,10))
ax[0, 0].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[0, 0].set_title('Imagen Original')
ax[0, 0].axis('off')
ax[0, 1].imshow(resul1, cmap='gray', vmin=0, vmax=255)
ax[0, 1].set_title("Filtro Passa-baixas {res}x{res}")
ax[0, 1].axis('off')
ax[1, 0].imshow(resul2, cmap='gray', vmin=0, vmax=255)
ax[1, 0].set_title('Filtro Laplaciano')
ax[1, 0].axis('off')
ax[1, 1].imshow(resul3, cmap='gray', vmin=0, vmax=255)
ax[1, 1].set_title('Filtro de Sobel')
ax[1, 1].axis('off')

Out[ ]: (-0.5, 511.5, 511.5, -0.5)
```

Imagen Original



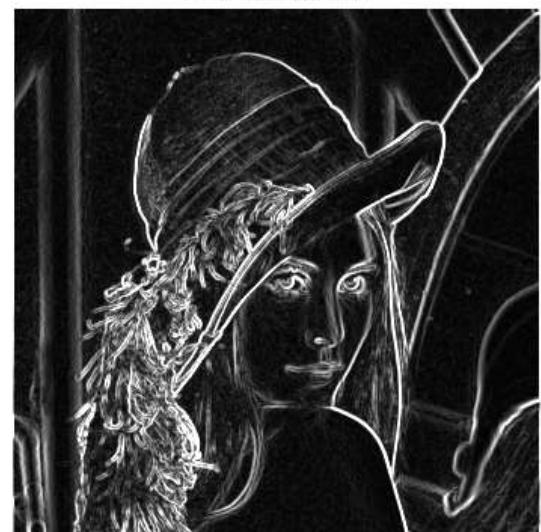
Filtro Passa-baixas 3x3



Filtro Laplaciano



Filtro de Sobel



- O Filtro Passa-baixas é um filtro de suavização, portanto a imagem obtida é ligeiramente borrada em relação à original.
- Já o Filtro Laplaciano é um filtro de aguçamento, ou seja, destaca as transições de intensidade de forma a aumentar a nitidez da imagem. No caso, este filtro pode ser

adicionado a imagem original, obtendo diretamente a imagem com suas bordas aguçadas, como foi realizado para obter a imagem acima.

- O Filtro de Sobel, assim como o Laplaciano, é um filtro de aguçamento, porém, seu uso geralmente visa obter não uma versão aguçada da imagem original (obtida adicionando a imagem filtrada com o Laplaciano à imagem original), mas uma versão da imagem com ênfase dos locais onde existem grandes transições de intensidade. Sua aplicação é voltada para ressaltar as bordas de objetos numa imagem e como pré-processamento para aplicações de detecção de bordas.
-

4)

Aplique uma filtragem de high-boost com $k = 1,5$ e filtro de suavização de média aritmética simples de dimensão 5x5 na imagem Fig3.40(a).jpg.

Depois aplique sobre a mesma imagem um filtro laplaciano (adicionando a imagem original ao resultado) isotrópico para rotações de 45° .

Exiba as imagens e compare os resultados obtidos por cada método e com a imagem original.

Resposta:

```
In [28]: #####
# Filtro High-Boost
...
Aplica um Filtro High-Boost em uma imagem
* inputs: imagem, resolucao do filtro e constante de multiplicacao da mascara
* output: imagem de entrada modificada
* pre-condicao: imagem valida (diferente de None), resolucao com valor
    valido (impar)
* pos-condicao: imagem de saida foi alterada de acordo com a mascara
...
def filtro_HighBoost(image, res = 3, k = 1):
    # Primeiro aplica um filtro de borramento:
    img_borrada = filtro_PassaBaixas(image, res)

    # Aplicando o Filtro High-Boost
    resul = image + k*(image - img_borrada)
    return resul

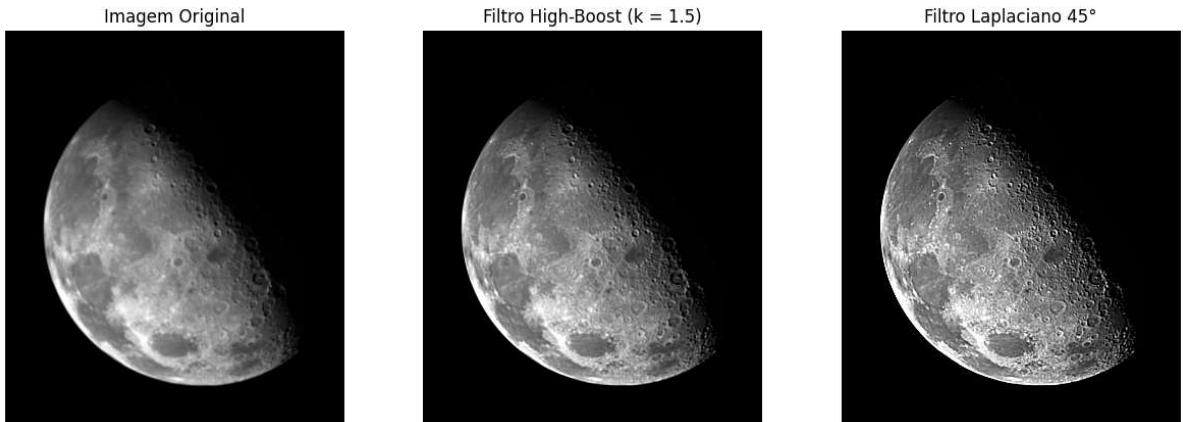
#####
# Filtro Laplaciano com incrementos de 45°
...
Aplica um Filtro Laplaciano de 45° em uma imagem
* inputs: imagem
* output: imagem de entrada modificada
* pre-condicao: imagem valida (diferente de None)
* pos-condicao: imagem de saida foi alterada de acordo com a mascara
...
def filtro_Laplaciano45(image):
    # Filtro Laplaciano com adicao da imagem original
    kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])
```

```
resul = convEspacial(image, kernel)
return resul
```

```
In [38]: # Lendo as imagens:
image = plt.imread(list_path+'Fig3.40(a).jpg')
image = image.astype(np.float32) # ou np.float64

# Processando a imagem:
resul1 = filtro_HighBoost(image, res = 5, k = 1.5)
resul2 = filtro_Laplaciano45(image)
```

```
In [41]: # Exibir imagem
i = 0
fig, ax = plt.subplots(1, 3, figsize=(15,5))
ax[i].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[i].set_title('Imagen Original')
ax[i].axis('off')
i = i+1
ax[i].imshow(resul1, cmap='gray', vmin=0, vmax=255)
ax[i].set_title('Filtro High-Boost (k = 1.5)')
ax[i].axis('off')
i = i+1
ax[i].imshow(resul2, cmap='gray', vmin=0, vmax=255)
ax[i].set_title('Filtro Laplaciano 45°')
ax[i].axis('off')
```



- Como ambos filtros (High-boost e Laplaciano 45°) são filtros de aguçamento, obtemos imagens com maior nitidez quando aplicados.
- O Filtro High-boost, como está sendo utilizado com $k = 1.5$, aumenta a nitidez da imagem, evidenciando as crateras e outros elementos, mas ainda fica um pouco distante do aumento brusco do Filtro Laplaciano de 45°. Sua vantagem é justamente devido ao fator k , que faz com que a nitidez possa ser amplificada aumentando este fator até que seja obtido o resultado desejado.
- O Filtro Laplaciano 45°, ao adicionar as derivadas nas direções diagonais, acentua ainda mais as regiões da imagem onde há transições de intensidade. Comparando a imagem original é possível localizar com grandíssima facilidade as crateras e outros elementos presentes na imagem da Lua.

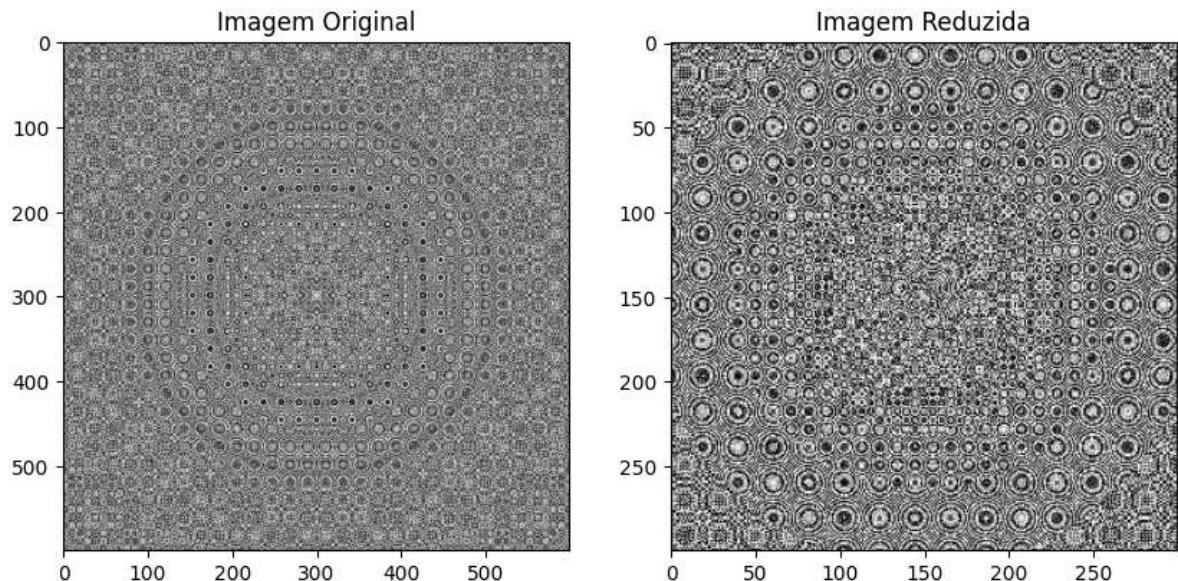
Reduza a imagem frexp_1.png eliminando alternadamente as linhas e colunas.

Compare a imagem original com a reduzida. O que aconteceu com ela? Que procedimento você poderia aplicar para reduzir esse efeito?

Aplique a solução imaginada para evitar esse efeito na imagem reduzida. Compare a nova imagem com as outras duas e avalie o resultado.

Resposta:

```
In [ ]: # Lendo as imagens:  
# Necessario multiplicar por 255 as imagens do tipo PNG  
image = 255*plt.imread(list_path+'frexp_1.png')  
image = image.astype(np.float32) # ou np.float64  
  
# Processando a imagem:  
resul1 = image[0::2, 1::2]  
  
# Exibir imagem  
fig, ax = plt.subplots(1, 2, figsize=(10,5))  
ax[0].imshow(image, cmap='gray', vmin=0, vmax=255)  
ax[0].set_title('Imagen Original')  
ax[1].imshow(resul1, cmap='gray', vmin=0, vmax=255)  
ax[1].set_title('Imagen Reduzida')  
  
Out[ ]: Text(0.5, 1.0, 'Imagen Reduzida')
```

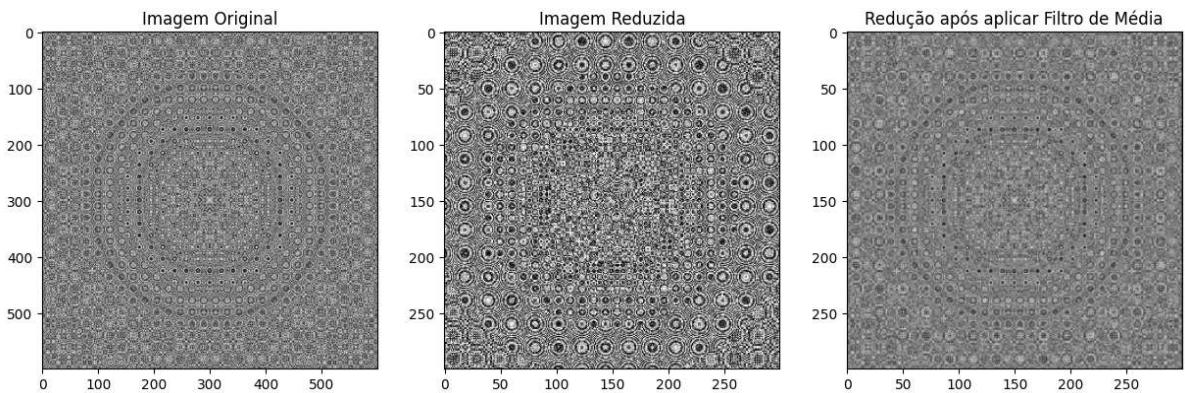


- A imagem reduzida apresenta múltiplos artefatos não contidos na imagem original, característico do efeito de "aliasing". Esse efeito pode surgir quando se faz downsampling (subamostragem) de uma imagem.
- Uma boa solução é aplicar um filtro de média aritmética (também chamado filtro passa-baixas) antes de realizar a redução da imagem.

```
In [ ]: # Correcao  
# Aplicar o Filtro Passa-Baixas (ou Filtro de Media) antes de diminuir a imagem  
resul2_ = filtro_PassaBaixas(image)  
resul2 = resul2_[0::2, 1::2]
```

```
# Exibir imagem
fig, ax = plt.subplots(1, 3, figsize=(15,5))
ax[0].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[0].set_title('Imagen Original')
ax[1].imshow(resul1, cmap='gray', vmin=0, vmax=255)
ax[1].set_title('Imagen Reduzida')
ax[2].imshow(resul2.astype('float16'), cmap='gray', vmin=0, vmax=255)
ax[2].set_title('Redução após aplicar Filtro de Média')
```

Out[]: Text(0.5, 1.0, 'Redução após aplicar Filtro de Média')



- Como pode ser observado na imagem acima, aplicando o filtro de média aritmética antes de realizar a redução da imagem faz com que não ocorra o surgimento de artefatos na imagem. Porém, como foi aplicado o filtro de média, a imagem apresenta o borramento característico deste filtro.

6)

Faça uma rotina que implemente, no domínio da frequência, um filtro passa baixa de Butterworth de ordem n e frequência de corte D_0 . Aplique esse filtro com $n = 1$ sobre a imagem lena.tif para dois valores de frequência de corte. Aplique novamente o filtro sobre a mesma imagem e mesmas frequências de corte, mas com $n = 8$. Comente os resultados.

Resposta:

```
In [17]: #####
# Filtro Butterworth
...
Aplica um Filtro Butterworth em uma imagem
* inputs: imagem, frequencia de corte e ordem do filtro
* output: imagem de entrada modificada
* pre-condicao: imagem valida (diferente de None)
* pos-condicao: imagem de saida foi alterada de acordo com o filtro especificado
...
def butterworthFilter(image, D0 = 1, n = 1):
    P, Q = image.shape
    H = np.zeros((2*P, 2*Q))

    # Calculando o filtro:
    for u in range(2*P):
        for v in range(2*Q):
```

```

D = np.sqrt((u-P)**2 + (v-Q)**2)
H[u, v] = 1 / (1 + ((D/D0)**(2*n)))

# Preenche a imagem com zeros à direita e abaixo
image_filled = np.zeros((2*P, 2*Q))
image_filled[0:P, 0:Q] = image

# Passa a imagem para o domínio da frequência
# fft2 = 2D Fast Fourier Transformation
# fftshift = Coloca a componente de frequência zero no centro da imagem
F = np.fft.fftshift(np.fft.fft2(image_filled))

# Multiplica ponto a ponto a imagem no domínio da frequência pelo filtro
G = F*H

# Retorna o resultado para o domínio espacial
resul_filled = np.real(np.fft.ifft2(np.fft.fftshift((G))))
resul = resul_filled[0:P, 0:Q]

return resul

```

In [18]: # Lendo as imagens:

```

image = plt.imread(list_path+'lena.tif')
image = image.astype(np.float32) # ou np.float64

```

In [26]: D01 = 70
D02 = 160

```

# Filtro de ordem 1
F_n1_low = butterworthFilter(image, D0 = D01, n = 1)
F_n1_high = butterworthFilter(image, D0 = D02, n = 1)

# Filtro de ordem 8
F_n8_low = butterworthFilter(image, D0 = D01, n = 8)
F_n8_high = butterworthFilter(image, D0 = D02, n = 8)

```

In [27]: fig, ax = plt.subplots(2, 3, figsize=(15,10))
i = 0

```

ax[0, i].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[0, i].set_title('Imagen Original')
ax[0, i].axis('off')
i = i+1
ax[0, i].imshow(F_n1_low, cmap='gray', vmin=0, vmax=255)
ax[0, i].set_title(f"BLPF D0 = {D01}; n = 1")
ax[0, i].axis('off')
i = i+1
ax[0, i].imshow(F_n1_high, cmap='gray', vmin=0, vmax=255)
ax[0, i].set_title(f"BLPF D0 = {D02}; n = 1")
ax[0, i].axis('off')

i = 0
ax[1, i].imshow(image, cmap='gray', vmin=0, vmax=255)
ax[1, i].set_title('Imagen Original')
ax[1, i].axis('off')
i = i+1
ax[1, i].imshow(F_n8_low, cmap='gray', vmin=0, vmax=255)
ax[1, i].set_title(f"BLPF D0 = {D01}; n = 8")
ax[1, i].axis('off')
i = i+1
ax[1, i].imshow(F_n8_high, cmap='gray', vmin=0, vmax=255)
ax[1, i].set_title(f"BLPF D0 = {D02}; n = 8")
ax[1, i].axis('off')

```

Out[27]: (-0.5, 511.5, 511.5, -0.5)



- A partir dos resultados é possível observar que o aumento da frequência de corte provocou um menor borramento e maior porcentagem de potência em relação à imagem original, o que é característico dos filtros passa-baixas.
- Além disso, o aumento da ordem do filtro da ordem 1 para ordem 8, causou o efeito de ringing, característico dos filtro de Butterworth de ordens maiores. Esse efeito é muito perceptível na imagem para frequência de corte 70 e para a maior frequência é bem perceptível apenas nas bordas da imagem.

7)

Para as imagens ruidosa1.jpg e ruidosa2.jpg, aplique os filtros média 5x5, mediana 5x5 e o filtro adaptativo de mediana (indo de dimensão 3x3 até a dimensão 11x11) para tratar o ruído. Compare com a imagem original.jpg calculando-se a PSNR (Peak Signal to Noise Ratio) segundo a equação:

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \quad (1)$$

$$PSNR = 20 \log_{10} \left(\frac{MAX_I}{MSE} \right) \quad (2)$$

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2 \quad (3)$$

onde $I(i, j)$ e $K(i, j)$ são as imagens original e a ruidosa, respectivamente, MAX é o maior valor possível de nível de cinza, sendo $MAX = 255$, $m \cdot n$ é o número de pixels da imagem. Usar \log_{10} para calcular a $PSNR$.

Compare os resultados e discuta.

Resposta:

```
In [7]: #####
# Filtro de Media Aritmetica
...
Aplica um filtro de media aritmetica a uma imagem
* inputs:
  * imagem
  * resolucao do filtro
  * tipo de padding para ser utilizado:
    * 0 => Padding de zeros
    * 1 => Padding reflexivo
* output: imagem de entrada modificada
* pre-condicao: imagem valida (diferente de None) e resolucao valida
  (resolucao impar)
* pos-condicao: imagem de saida foi alterada de acordo com o filtro de
  resolucao especificada
...

def filtroMedia(image, res = 3, padding_type = 0):
    if res%2 != 1:
        print("Resolucao invalida! \nA resolucao do filtro deve ser um valor impar!")
        return -1

    # VALIDADAS AS CONDICOES, INICIA-SE A OPERACAO DE FATO
    # Cria uma copia da imagem original para alocar a imagem resultante
    lin, col = image.shape
    resultado = np.zeros((lin, col))

    # Fazendo o padding de zeros na imagem
    padding_size = int((res-1)/2)
    if padding_type == 0:
        image_padded = np.zeros((lin+2*padding_size, col+2*padding_size))
        image_padded[padding_size:lin+padding_size, padding_size:col+padding_size] = image
    elif padding_type == 1:
        image_padded = np.pad(image, padding_size, mode='reflect')

    # Percorre a imagem com padding aplicando a transformacao
    lin, col = image_padded.shape

    # Fator potencia
    fator = res**2

    for i in range(lin-2*padding_size):
        for j in range(col-2*padding_size):
            sub_area = image_padded[i:i+res, j:j+res]
            resultado[i, j] = int((np.sum(sub_area))/fator)

    return resultado

#####
# Filtro de Mediana
...
Aplica um filtro de mediana a uma imagem
* inputs:
  * imagem
  * resolucao do filtro
  * tipo de padding para ser utilizado:
    * 0 => Padding de zeros
    * 1 => Padding reflexivo
```

```

    * output: imagem de entrada modificada
    * pre-condicao: imagem valida (diferente de None) e resolucao valida
        (resolucao impar)
    * pos-condicao: imagem de saida foi alterada de acordo com o filtro de
        resolucao especificada
    ...

def filtroMediana(image, res = 3, padding_type = 0):
    if res%2 != 1:
        print("Resolucao invalida! \nA resolucao do filtro deve ser um valor impar!")
        return -1

    # VALIDADAS AS CONDICOES, INICIA-SE A OPERACAO DE FATO
    # Cria uma copia da imagem original para alocar a imagem resultante
    lin, col = image.shape
    resultado = np.zeros((lin, col))

    # Fazendo o padding de zeros na imagem
    padding_size = int((res-1)/2)
    if padding_type == 0:
        image_padded = np.zeros((lin+2*padding_size, col+2*padding_size))
        image_padded[padding_size:lin+padding_size, padding_size:col+padding_size] = image
    elif padding_type == 1:
        image_padded = np.pad(image, padding_size, mode='reflect')

    # Percorre a imagem com padding aplicando a transformacao
    lin, col = image_padded.shape

    for i in range(lin-2*padding_size):
        for j in range(col-2*padding_size):
            sub_area = image_padded[i:i+res, j:j+res]
            resultado[i, j] = int(np.median(sub_area, axis=None))

    return resultado

#####
# Filtro de Mediana Adaptativa
///

Aplica um filtro de mediana adaptativa a uma imagem
    * inputs:
        * imagem
        * resolucao minima do filtro
        * resolucao maxima do filtro
        * tipo de padding para ser utilizado:
            * 0 => Padding de zeros
            * 1 => Padding reflexivo
    * output: imagem de entrada modificada
    * pre-condicao: imagem valida (diferente de None) e resolucoes validas
        (resolucao impar)
    * pos-condicao: imagem de saida foi alterada de acordo com o filtro de
        resolucoes especificadas
    ...

def filtroMedianaAdap(image, res_min = 3, res_max = 5, padding_type = 0):
    if res_min%2 != 1 or res_max%2 != 1:
        print("Resolucao invalida! \nA resolucao do filtro deve ser um valor impar!")
        return -1

    # VALIDADAS AS CONDICOES, INICIA-SE A OPERACAO DE FATO
    # Cria uma copia da imagem original para alocar a imagem resultante
    lin, col = image.shape
    resultado = np.zeros((lin, col))

    # Fazendo o padding de zeros na imagem
    padding_size = int((res_max-1)/2)

```

```

if padding_type == 0:
    image_padded = np.zeros((lin+2*padding_size, col+2*padding_size))
    image_padded[padding_size:lin+padding_size, padding_size:col+padding_size] = image
elif padding_type == 1:
    image_padded = np.pad(image, padding_size, mode='reflect')

# Percorre a imagem com padding aplicando a transformacao
lin, col = image_padded.shape

# for implementado com base nas coordenadas da imagem dentro da imagem com padding
for i in range(padding_size, lin-padding_size):
    for j in range(padding_size, col-padding_size):
        res = res_min
        while(True):
            shift = int(res/2)
            sub_area = image_padded[i-shift:i-shift+res, j-shift:j-shift+res]

            sa_median = np.median(sub_area, axis=None)
            sa_min = np.min(sub_area)
            sa_max = np.max(sub_area)
            if (sa_median - sa_min) > 0 and (sa_median - sa_max) < 0:
                # Estagio B
                if (image_padded[i,j] - sa_min) > 0 and (image_padded[i,j] - sa_max) < 0:
                    resultado[i-padding_size, j-padding_size] = image_padded[i, j]
                    break
                else:
                    resultado[i-padding_size, j-padding_size] = sa_median
                    break
            else:
                if res < res_max:
                    res = res+2
                else:
                    resultado[i-padding_size, j-padding_size] = sa_median
                    break
        return resultado

#####
# Calculo de PSNR
'''

Calcula a relacao sinal-ruido de pico (PSNR - Peak Signal to Noise Ratio) entre
duas imagens
* inputs:
    * imagem original (sem ruido)
    * imagem para comparacao (ruidosa ou tratada por filtro)
* output: valor da relacao sinal-ruido de pico
* pre-condicao: imagens validas (diferente de None) e diferentes
* pos-condicao: valor da relacao sinal-ruido de pico foi calculado
'''

def PSNR(original, ruidosa):
    M, N = original.shape

    dif = original - ruidosa
    MSE = np.sum(dif**2)/(M*N)

    PSNR = 20*np.log10(255/np.sqrt(MSE))

    return PSNR

```

In [8]:

```

# Lendo as imagens:
ruidosa1 = plt.imread(list_path+'ruidosa1.tif')
ruidosa1 = ruidosa1.astype(np.float32) # ou np.float64
ruidosa2 = plt.imread(list_path+'ruidosa2.tif')
ruidosa2 = ruidosa2.astype(np.float32) # ou np.float64

```

Com padding de zeros:

In [9]:

```
# Usando padding de zeros
result1_1 = filtroMedia(ruidosa1, res = 5)
result1_2 = filtroMediana(ruidosa1, res = 5)
result1_3 = filtroMedianaAdap(ruidosa1, res_min = 3, res_max = 11)

result2_1 = filtroMedia(ruidosa2, res = 5)
result2_2 = filtroMediana(ruidosa2, res = 5)
result2_3 = filtroMedianaAdap(ruidosa2, res_min = 3, res_max = 11)
```

In []:

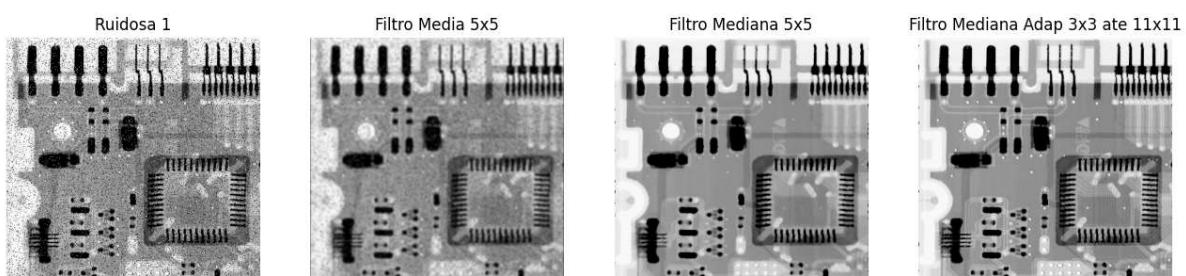
```
i = 0
fig, ax0 = plt.subplots(1, 4, figsize=(16,4))
ax0[i].imshow(ruidosa1, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Ruidosa 1')
ax0[i].axis('off')
i = i+1
ax0[i].imshow(result1_1, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Filtro Media 5x5')
ax0[i].axis('off')
i = i+1
ax0[i].imshow(result1_2, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Filtro Mediana 5x5')
ax0[i].axis('off')
i = i+1
ax0[i].imshow(result1_3, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Filtro Mediana Adap 3x3 ate 11x11')
ax0[i].axis('off')

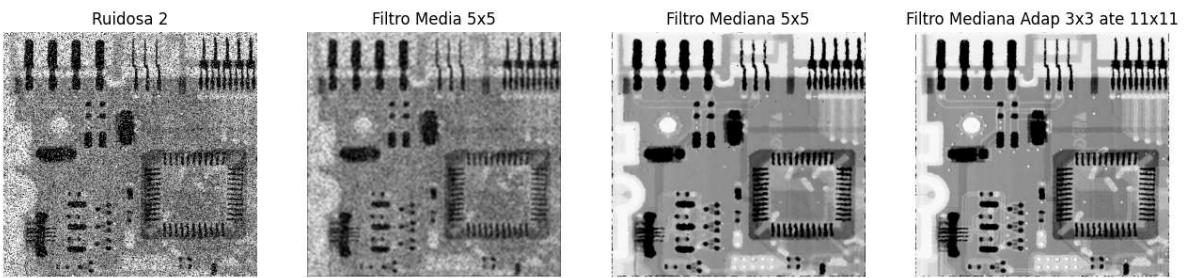
i = 0
fig1, ax1 = plt.subplots(1, 4, figsize=(16,4))
ax1[i].imshow(ruidosa2, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Ruidosa 2')
ax1[i].axis('off')
i = i+1
ax1[i].imshow(result2_1, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Filtro Media 5x5')
ax1[i].axis('off')
i = i+1
ax1[i].imshow(result2_2, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Filtro Mediana 5x5')
ax1[i].axis('off')
i = i+1
ax1[i].imshow(result2_3, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Filtro Mediana Adap 3x3 ate 11x11')
ax1[i].axis('off')
fig.suptitle('Filtros com padding de zeros', fontsize=16)
```

Out[]:

Text(0.5, 0.98, 'Filtros com padding de zeros')

Filtros com padding de zeros





- Ao aplicar os filtros com o padding de zeros (solução escolhida para implementação do padding), notou-se que as bordas da imagem ficaram bem prejudicadas, independentemente do método utilizado, devido ao ruído das imagens conter o ruído do tipo pimenta.
- Então, optou-se por comparar os resultados obtidos também ao aplicar o padding reflexivo, para tentar minimizar os ruídos apresentados nas bordas após a filtragem.

Com padding reflexivo:

```
In [11]: # Usando padding reflexivo
result1_4 = filtroMedia(ruidosa1, res = 5, padding_type = 1)
result1_5 = filtroMediana(ruidosa1, res = 5, padding_type = 1)
result1_6 = filtroMedianaAdap(ruidosa1, res_min = 3, res_max = 11, padding_type = 1)

result2_4 = filtroMedia(ruidosa2, res = 5, padding_type = 1)
result2_5 = filtroMediana(ruidosa2, res = 5, padding_type = 1)
result2_6 = filtroMedianaAdap(ruidosa2, res_min = 3, res_max = 11, padding_type = 1)
```

```
In [ ]: i = 0
fig, ax0 = plt.subplots(1, 4, figsize=(16,4))
ax0[i].imshow(ruidosa1, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Ruidosa 1')
ax0[i].axis('off')
i = i+1
ax0[i].imshow(result1_4, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Filtro Media 5x5')
ax0[i].axis('off')
i = i+1
ax0[i].imshow(result1_5, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Filtro Mediana 5x5')
ax0[i].axis('off')
i = i+1
ax0[i].imshow(result1_6, cmap='gray', vmin=0, vmax=255)
ax0[i].set_title('Filtro Mediana Adap 3x3 ate 11x11')
ax0[i].axis('off')

i = 0
fig1, ax1 = plt.subplots(1, 4, figsize=(16,4))
ax1[i].imshow(ruidosa2, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Ruidosa 2')
ax1[i].axis('off')
i = i+1
ax1[i].imshow(result2_4, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Filtro Media 5x5')
ax1[i].axis('off')
i = i+1
ax1[i].imshow(result2_5, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Filtro Mediana 5x5')
ax1[i].axis('off')
i = i+1
```

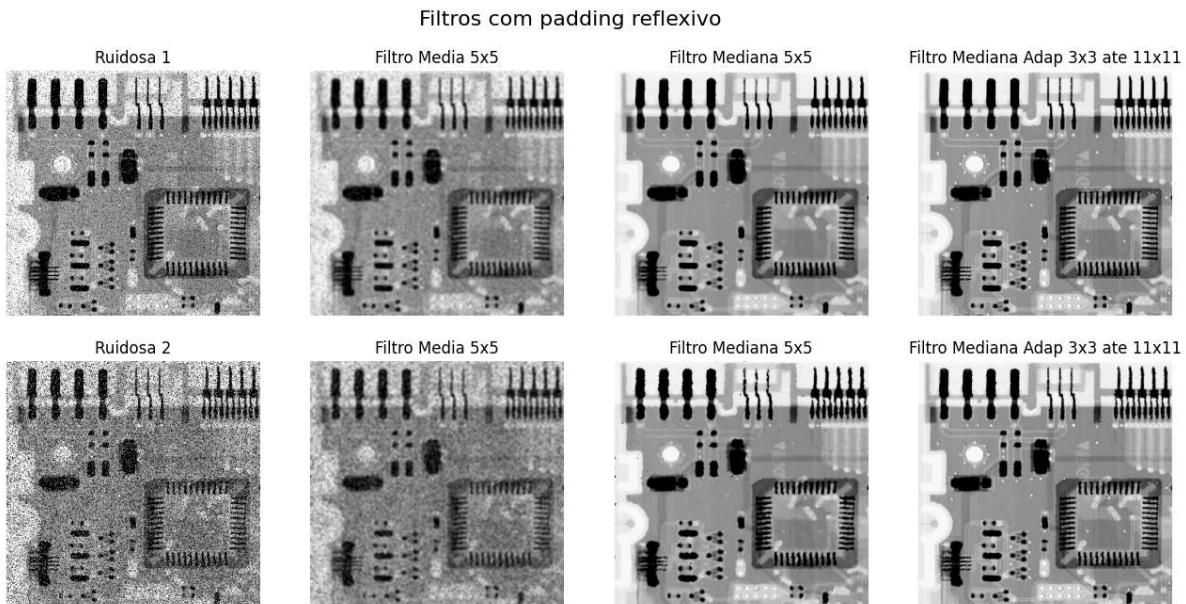
```

ax1[i].imshow(resul2_6, cmap='gray', vmin=0, vmax=255)
ax1[i].set_title('Filtro Mediana Adap 3x3 ate 11x11')
ax1[i].axis('off')
fig.suptitle('Filtros usando ', fontsize=16)

fig.suptitle('Filtros com padding reflexivo', fontsize=16)

```

Out[]: Text(0.5, 0.98, 'Filtros com padding reflexivo')



- Observando os resultados da filtragem com padding reflexivo, houve grande melhora em relação às bordas da imagem comparado aos resultados obtidos com a filtragem usando padding de zeros.

PSNR:

```

In [42]: original = plt.imread(list_path+'original.tif')[::,:,0]
original = original.astype(np.float32) # ou np.float64

PSNR1_1 = PSNR(original, resul1_1)
PSNR1_2 = PSNR(original, resul1_2)
PSNR1_3 = PSNR(original, resul1_3)

PSNR2_1 = PSNR(original, resul2_1)
PSNR2_2 = PSNR(original, resul2_2)
PSNR2_3 = PSNR(original, resul2_3)

PSNR1_4 = PSNR(original, resul1_4)
PSNR1_5 = PSNR(original, resul1_5)
PSNR1_6 = PSNR(original, resul1_6)

PSNR2_4 = PSNR(original, resul2_4)
PSNR2_5 = PSNR(original, resul2_5)
PSNR2_6 = PSNR(original, resul2_6)

print(f"-----")
print(f"Filtros com padding de zeros")
print(f"-----")
print(f"Ruidosa 1:")
print(f"PSNR Filtro Media = \t\t {PSNR1_1:.4f}")
print(f"PSNR Filtro Mediana = \t\t {PSNR1_2:.4f}")
print(f"PSNR Filtro Mediana Adaptativa = {PSNR1_3:.4f}")

```

```

print(f"-----")
print(f"Ruidosa 2:")
print(f"PSNR Filtro Media = \t\t {PSNR2_1:.4f}")
print(f"PSNR Filtro Mediana = \t\t {PSNR2_2:.4f}")
print(f"PSNR Filtro Mediana Adaptativa = {PSNR2_3:.4f}")
print(f"-----")

print(f"\n-----")
print(f"Filtros com padding reflexivo")
print(f"-----")
print(f"Ruidosa 1:")
print(f"PSNR Filtro Media = \t\t {PSNR1_4:.4f}")
print(f"PSNR Filtro Mediana = \t\t {PSNR1_5:.4f}")
print(f"PSNR Filtro Mediana Adaptativa = {PSNR1_6:.4f}")

print(f"-----")
print(f"Ruidosa 2:")
print(f"PSNR Filtro Media = \t\t {PSNR2_4:.4f}")
print(f"PSNR Filtro Mediana = \t\t {PSNR2_5:.4f}")
print(f"PSNR Filtro Mediana Adaptativa = {PSNR2_6:.4f}")
print(f"-----")

```

Filtros com padding de zeros

Ruidosa 1:

```

PSNR Filtro Media =      20.0183
PSNR Filtro Mediana =   24.8198
PSNR Filtro Mediana Adaptativa = 30.2882

```

Ruidosa 2:

```

PSNR Filtro Media =      15.9323
PSNR Filtro Mediana =   19.8398
PSNR Filtro Mediana Adaptativa = 22.8007

```

Filtros com padding reflexivo

Ruidosa 1:

```

PSNR Filtro Media =      20.7317
PSNR Filtro Mediana =   25.6450
PSNR Filtro Mediana Adaptativa = 32.6215

```

Ruidosa 2:

```

PSNR Filtro Media =      16.2629
PSNR Filtro Mediana =   22.0335
PSNR Filtro Mediana Adaptativa = 26.5600

```

- Analisando a equação de cálculo da PSNR, vemos que quanto maior a similaridade entre a imagem original e a imagem filtrada (ou imagem ruidosa), ou seja, quanto menor a diferença entre ambas, maior o valor da PSNR.

$$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \quad (4)$$

$$\lim_{[I(i,j)-K(i,j)] \rightarrow 0} MSE = 0 \quad (5)$$

$$PSNR = 20 \log_{10} \left(\frac{MAX_I}{MSE} \right) \quad (6)$$

$$\lim_{MSE \rightarrow 0} PSNR = \infty \quad (7)$$

- Primeiramente, analisando os resultados obtidos em relação ao tipo de filtro utilizado:
 - Em todos os casos, o filtro de media apresentou o pior resultado seguido do filtro de mediana, e o filtro de mediana adaptativa apresentou o melhor resultado.
 - Também é possível observar que a performance de todos os filtros é afetada pelo aumento de ruído da imagem ruidosa2 em relação a imagem ruidosa1.
 - Já em relação à escolha do tipo de padding, para todos os filtros e em ambas imagens, o uso do padding reflexivo melhorou a relação sinal-ruído de pico em relação ao uso do padding de zeros. Devido a presença de um ruído do tipo pimenta nas imagens, o padding de zeros atrapalha na remoção dos ruídos nas bordas das imagens.
-

Perguntas:

1) Por que na equalização de histograma de uma imagem não é possível obter uma distribuição perfeitamente uniforme?

Resposta: O resultado da equalização distribui os valores uniformemente, porém, em valores não inteiros, por exemplo:

- Para uma imagem com 8 níveis de intensidade ($L = 8$), onde r_k são os valores dos pixels e n_k é o número de pixels na imagem para cada intensidade:

r_k	n_k	$p_r(r_k)$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

(8)

- São remapeados para:

r_k	s_k
$r_0 = 0$	$s_0 = 1.33$
$r_1 = 1$	$s_1 = 3.08$
$r_2 = 2$	$s_2 = 4.55$
$r_3 = 3$	$s_3 = 5.67$
$r_4 = 4$	$s_4 = 6.23$
$r_5 = 5$	$s_5 = 6.65$
$r_6 = 6$	$s_6 = 6.86$
$r_7 = 7$	$s_7 = 7.00$

(9)

- Onde

$$s_k = (L - 1) \sum_{j=0}^k p_r \cdot r_j \quad (10)$$

- Porém, como as imagens digitais adotam somente valores inteiros, esses valores serão arredondados:

r_k	s_k
$r_0 = 0$	$s_0 = 1.33 \approx 1$
$r_1 = 1$	$s_1 = 3.08 \approx 3$
$r_2 = 2$	$s_2 = 4.55 \approx 5$
$r_3 = 3$	$s_3 = 5.67 \approx 6$
$r_4 = 4$	$s_4 = 6.23 \approx 6$
$r_5 = 5$	$s_5 = 6.65 \approx 7$
$r_6 = 6$	$s_6 = 6.86 \approx 7$
$r_7 = 7$	$s_7 = 7.00 \approx 7$

(11)

- Sendo assim, múltiplos valores são remapeados para um mesmo valor, tornando a distribuição não exatamente uniforme, porém mais próxima de uma distribuição uniforme que a original da imagem.
-

2) Olhando para o espectro de Fourier e sua respectiva imagem de origem, é possível fazer um paralelo entre, por exemplo, a localização de uma borda na imagem e a componente espectral que ela gerou? E o contrário, olhando somente para uma componente espectral, é possível dizer qual região da imagem a gerou (que pixels, suas localizações)?

Resposta: Olhando tanto para o espectro de Fourier e para a imagem é possível fazer um paralelo entre as componentes espetrais e quais bordas/mudanças bruscas de tonalidade na imagem as geraram, mas isso não é possível para todas as componentes espetrais.

Porém, não é possível relacionar uma componente espectral diretamente a regiões específicas de uma imagem sem analisar a imagem em conjunto.

3) Explique os três métodos apresentados no livro texto para estimar a função de degradação na tarefa de restauração de imagens.

Resposta: Todos estes métodos assumem que a imagem foi degradada por um processo linear e invariante no espaço. [Pág 245]

1. Estimativa pela observação da imagem

- Para estimar a função de degradação H , deve-se analisar diretamente subáreas da imagem que possam conter informações relevantes sobre a degradação (partes da imagem onde o sinal de degradação é fortemente percebido).
- A escolha da região deve ser tal que o ruído possa ser assumido como mínimo e ignorado na estimativa da função de degradação.
- No domínio da frequência, temos que a imagem ($F(u, v)$) degradada por uma função de degradação ($H(u, v)$) é dada por:

$$G(u, v) = F(u, v) \cdot H(u, v) \quad (12)$$

- Para uma subárea da imagem degradada $G_s(u, v)$, assumindo que uma estimativa não degradada desta subárea seja $\hat{F}_s(u, v)$, então a função de degradação desta

subárea da imagem pode ser representada por:

$$H_s(u, v) = \frac{G_s(u, v)}{\hat{F}_s(u, v)} \quad (13)$$

- Essa função de degradação da subárea é usada para deduzir a função de degradação da imagem completa $H(u, v)$, assumindo a invariância no espaço.
- Por exemplo, se a imagem degradada apresenta um borramento, é possível aproximar $\hat{F}_s(u, v)$ aplicando uma filtragem de aguçamento e então estimar a função de degradação da subárea como a razão entre essa aproximação e a subárea da imagem degradada.

2. Estimativa por experimentação

- Se um equipamento similar ao que foi usado na aquisição da imagem degradada estiver disponível é possível obter uma estimativa da função de degradação.
- Testando múltiplas configurações do sistema, ao obter uma imagem com degradação similar a imagem que deseja-se restaurar, utiliza-se essas configurações para obter uma imagem de um impulso (pequeno ponto de luz). Já que a Transformada de Fourier do impulso é uma constante, é possível obter $H(u, v)$:

$$H(u, v) = \frac{G(u, v)}{A} \quad (14)$$

- Onde A é a constante que descreve a intensidade do impulso e $G(u, v)$ é a Transformada de Fourier da imagem degradada.

3. Estimativa por modelamento

- Essa abordagem utilizad modelos para estimar a função de degradação, por exemplo, o modelo de degradação proposto por Hufnagel e Stanley para turbulência atmosférica:

$$H(u, v) = e^{-k(u^2+v^2)^{5/6}} \quad (15)$$

- Onde k é uma constante que depende da natureza da turbulência.
- Além disso, é possível deduzir alguns modelos matemáticos a partir de princípios da física.

4) Explique a problemática relacionada à filtragem inversa aplicada para restauração de imagens.

Resposta: A filtragem inversa obtêm uma aproximação da imagem original $\hat{F}_s(u, v)$, dividindo diretamente a imagem degradada $G(u, v)$ pela função de degradação $H(u, v)$:

$$\hat{F}_s(u, v) = \frac{G(u, v)}{H(u, v)} \quad (16)$$

Porém, temos que a imagem degradada é composta por:

$$G(u, v) = H(u, v) \cdot F(u, v) + N(u, v) \quad (17)$$

Onde $N(u, v)$ é o ruído presente na imagem degradada. Substituindo essa expressão na equação de filtragem inversa:

$$\hat{F}_s(u, v) = F(u, v) + \frac{N(u, v)}{H(u, v)} \quad (18)$$

Algumas problemáticas são que:

- Como a componente $N(u, v)$ não é conhecida, não é possível obter a imagem original não degradada (transformada inversa de Fourier de $F(u, v)$).
- Se a função de degradação possuir valores iguais a zero ou muito pequenos, a componente $\frac{N(u, v)}{H(u, v)}$ dominará a estimativa de $\hat{F}_s(u, v)$. Porém, esse problema pode ser remediado limitando-se as frequências a valores próximos à origem (já que o valor $H(0, 0)$ costuma ser o mais alto de $H(u, v)$ no domínio da frequência), aplicando um filtro para limitar essas frequências.