Carregar as bibliotecas

```
import numpy as np
from skimage import io, data, filters, util, color
from scipy import ndimage, stats
import matplotlib.pyplot as plt
from skimage.util import img_as_float, img_as_uint


def mapping1D(img):
    nimg = img_as_float(img)
    mmin = np.min(nimg)
    nimg = nimg - mmin
    mmax = np.max(nimg)
    return nimg/mmax


def mappingND(img):
    dims = img.ndim
    nimg = np.zeros_like(img, dtype='float')
    for i in range(img.shape[dims-1]):
      nimg[:,:,i] = mapping1D(img[:,:,i])
    return nimg
```

Questão 1

Modifique o código *NoiseSum*, disponível nos slides, que remove o ruído de n imagens ruidosas através da média das n as imagens. No lugar da média use a mediana. Para encontrar a mediana de um conjunto de elementos basta ordenar os dados e selecionar o elemento que se encontra no meio do conjunto de dados. Por exemplo, seja $A = [4, 6, 2, 9, 1, 3, 9]$, depois de ordenar o vetor, fica da seguinte forma $A = [1, 2, 3, 4, 6, 9, 9]$. A mediana é o elemento que se encontra no meio do conjunto, neste caso seria o número 4. Para ordenar um conjunto de elementos use a função *numpy.sort(dados, axis=- 1)*, onde *dados são os elementos que vão ser ordenados e axis especifica em qual das dimensões da matriz vai ser realizada a ordenação dos dados, 0 ordenada por linhas, 1 por colunas e 2 por profundidade.*

Logo, modifique seu código usando a função numpy.median(dados, axis) para encontrar a imagem "mediana".

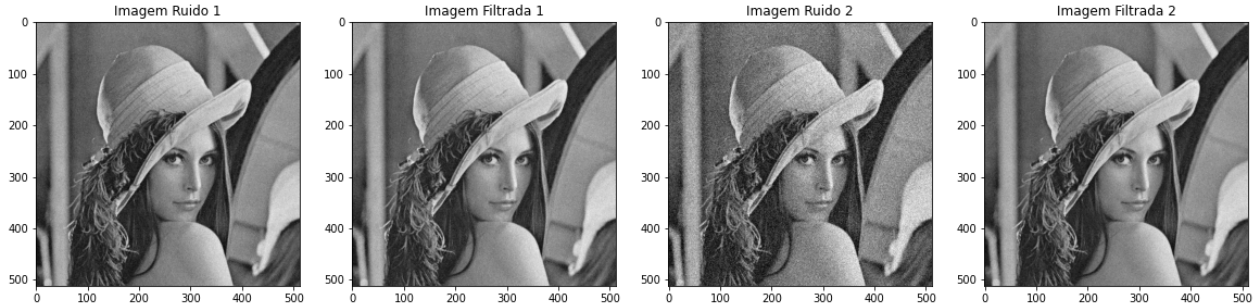
DICA: Salve as imagens ruidosas em uma matriz tridimensional para poder aplicar as funções sort e median. A variável figs da função NoiseSum é uma lista e não uma matriz

```
# função definada no slides da aula
def NoiseSum(img, n):
```

```
    nimg = np.zeros_like(img, dtype='float')
    figs = [None] * n
    for i in range(n):
        tmp = util.random_noise(img, 'gaussian')
        figs[i] = tmp
        nimg += figs[i]
    nimg /= n
    return figs, nimg

# usando a função sort
def NoiseSumMediana1(img, n):
    nimg = np.zeros_like(img, dtype='float')
    x,y=img.shape
    figs = np.zeros((n,x,y))
    for i in range(n):
        tmp = util.random_noise(img, 'gaussian')
        figs[i] = tmp
    figs = np.sort(figs,axis=0)
    newN = n//2
    nimg = figs[newN]
    print(nimg.shape)
    return figs, nimg


# usando a função median
def NoiseSumMediana2(img, n):
    nimg = np.zeros_like(img, dtype='float')
    x,y=img.shape
    figs = np.zeros((n,x,y))
    for i in range(n):
        tmp = util.random_noise(img, 'gaussian')
        figs[i] = tmp
    nimg = np.median(figs,axis=0)
    print(nimg.shape)
    print(figs.shape)
    return figs, nimg


lenna = io.imread('https://drive.google.com/uc?id=1k0FG4pT6WMjFeG-V63OBXNS7_CvfY5m_', as_g


figs1, nimg1 = NoiseSumMediana1(lenna, 10)
figs2, nimg2 = NoiseSumMediana2(lenna, 10)

f, ax = plt.subplots(1,4, figsize=(20,20))

ax[0].imshow(figs1[4,:,:], cmap='gray')
ax[0].set_title('Imagem Ruido 1')

ax[1].imshow(nimg1, cmap = 'gray')
ax[1].set_title('Imagem Filtrada 1')

ax[2].imshow(figs2[4,:,:], cmap='gray')
ax[2].set_title('Imagem Ruido 2')
```

```
ax[3].imshow(nimg2, cmap = 'gray')
ax[3].set_title('Imagem Filtrada 2')
```

```
(512, 512)
(512, 512)
(10, 512, 512)
Text(0.5, 1.0, 'Imagem Filtrada 2')
```



Questão 2

Mudar a cor do fundo (*background*) da imagem na escala de cinza, ela deve estar em tons de azul claro. Além da imagem em escala de cinza, é fornecida a imagem binária que é a máscara onde os fósforos estão representados pela cor branca e o fundo pela cor preta. Use operações aritméticas para executar o processo de modificação do fundo. A continuação são mostradas a imagem em escala de cinza e a sua respectiva máscara.

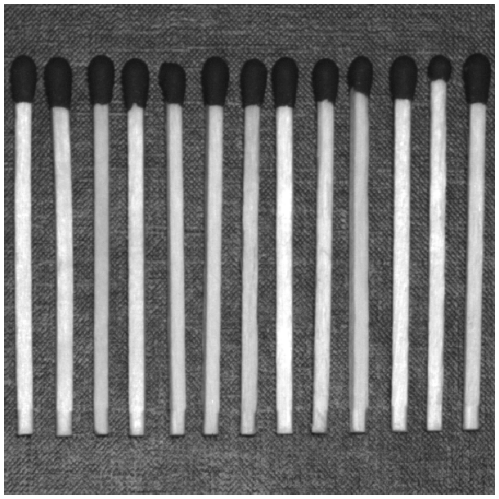
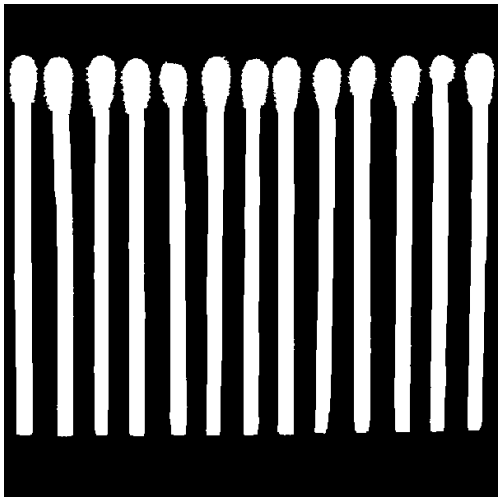


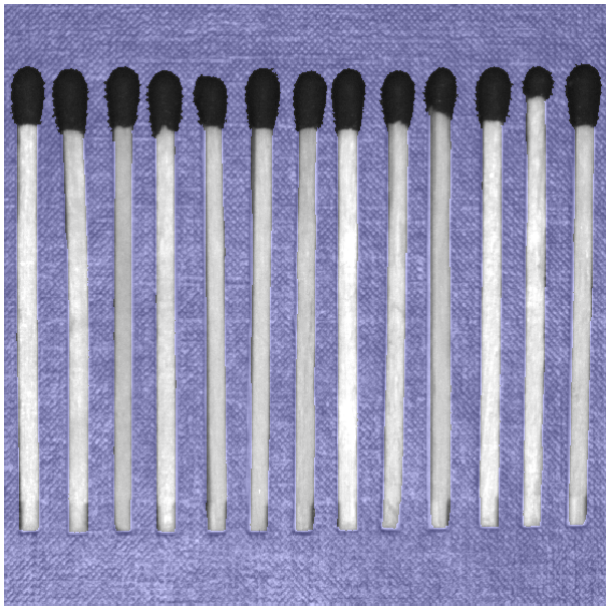
Imagem em escala de cinza


Máscara

DICA: para conseguir modificar a intensidade do fundo, incremente um valor inc nos canais vermelho e verde, e incremente um valor $2 * inc$ no canal azul. Preserve as intensidades originais dos fósforos. Não esqueça de converter a imagem para float, use a função

img_as_float. Depois de realizar as operações matemáticas, probablemente os valores dos pixels fiquem fora do intervalo válido $[0, 1]$. Use a função *Mapping* para normalizar os valores.

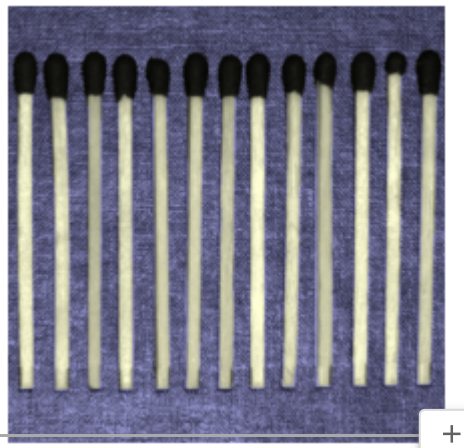
Figura a continuação mostra o resultado final. A imagem gerada deve ser colorida e o tecido (fundo da imagem) deve aparecer na cor azul claro.



Imágem resultante

```
def change_background(img, mask):
  print(img.shape)
  print(mask.shape)
  x,y=img.shape
  nimg=np.zeros((x,y,3))
  print(nimg.shape)
  nimg[:,:,0]=img
  nimg[:,:,1]=img
  nimg[:,:,2]=img
  nimg=img_as_float(nimg)

  nimg[:,:,0] = (((1+mask)%2) * (img * 0.2)) + (mask + img)
  nimg[:,:,1] = (((1+mask)%2) * (img * 0.2)) + (mask + img)
  nimg[:,:,2] = (((1+mask)%2) * (img * 0.9)) + (mask + img)


  nimg=mappingND(nimg)

  return nimg



img2 = io.imread('https://drive.google.com/uc?id=1V3OaLUh34dRCx-53PGsUYOMb1kZ1gqzo')
mask2 = io.imread('https://drive.google.com/uc?id=13NOF3oDNkmNtHCHmbX_9d0-AISPEmGZe')


nimg2 = change_background(img2, mask2)
plt.imshow(nimg2)
plt.axis('off')
```

```
(600, 600)
(600, 600)
(600, 600, 3)
(-0.5, 599.5, 599.5, -0.5)
```

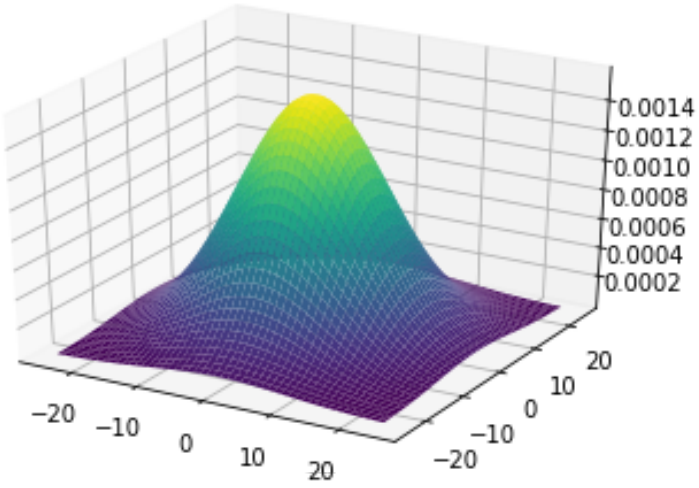


+ Código + Texto

Questão 3

A partir de duas imagens, gere o efeito que combina (*blend) duas imagens de forma tal que na imagem resultante, a parte central da primeira imagem aparece com maior intensidade e a medida que vai se afastando para as bordas vai perdendo intensidade. Um efeito inverso acontece com a segunda imagem, ela é mais intensa nas bordas da imagem, mas perde força na parte central.*

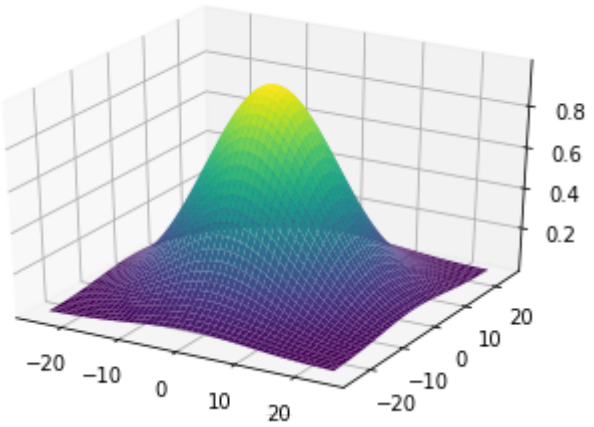
DICA: O peso atribuido para a primeira imagem tem o mesmo comportamento que uma função Gaussiana 2D,a parte central tem um peso maior e a medida que se afasta o peso vai diminuindo. Já a segunda imagem tem um comportamento inverso.



Função Gaussiana 2D

```
import math
def gaussian_mask(shape=(3,3), sigma=0.5):
    m,n = [(ss-1.)/2. for ss in shape]
    y,x = np.ogrid[-m:m+1,-n:n+1]
```

```
    h = (np.exp(-(x**2 + y**2) / (2*sigma**2)))
    sumh = h.sum()
    return h


shape = (50,50)
lin, col = [(ss-1.)/2. for ss in shape]
h2 = gaussian_mask(shape, sigma=10)

x, y = np.ogrid[-lin:lin+1, -col:col+1]
ax = plt.axes(projection='3d')
ax.plot_surface(x,y,h2, rstride=1, cstride=1, cmap='viridis', edgecolor='none');
```




```
def blend(img1=None, img2=None, sigma = 10):
  nimg = np.zeros_like(img1,dtype='float')
  x,y,c = img1.shape
  mask = gaussian_mask((x,y),sigma)
  img1aux=img_as_float(img1)
  img2aux=img_as_float(img2)

  nimg[:,:,0] = img1aux[:,:,0]* mask + img2aux[:,:,0]* (1-mask)
  nimg[:,:,1] = img1aux[:,:,1]* mask + img2aux[:,:,1]* (1-mask)
  nimg[:,:,2] = img1aux[:,:,2]* mask + img2aux[:,:,2]* (1-mask)

  return nimg


lenna = io.imread('https://drive.google.com/uc?id=1k0FG4pT6WMjFeG-V63OBXNS7_CvfY5m_', as_g
clown = io.imread('https://drive.google.com/uc?id=15yt6Tt5liol_jKWbkfcNXV-SoeaY706s', as_g
clown = color.rgba2rgb(clown)


img3 = blend(lenna, clown, 100)
plt.imshow(img3, cmap='gray')
plt.axis('off')
```

```
(-0.5, 511.5, 511.5, -0.5)
```

✓ 0s conclusão: 14:55