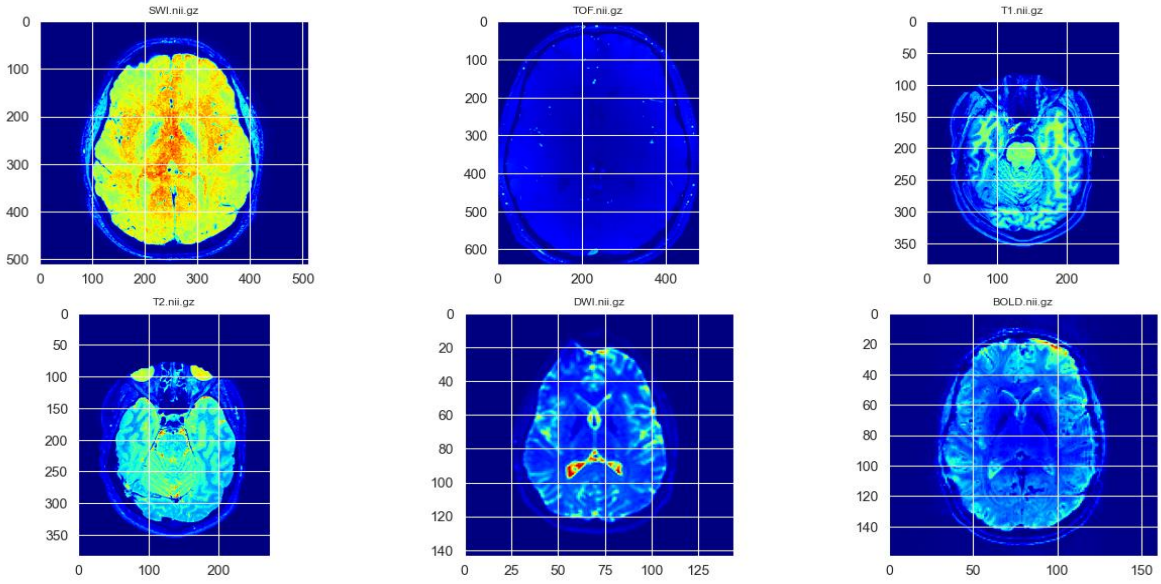
Bishop's university(Medical Imaging)
(Dr Russell Butler – Summer 2022)
Members:
Razieh Shahsavar (002341606) Maryam Bayatzadeh(002338161)

Assigment1

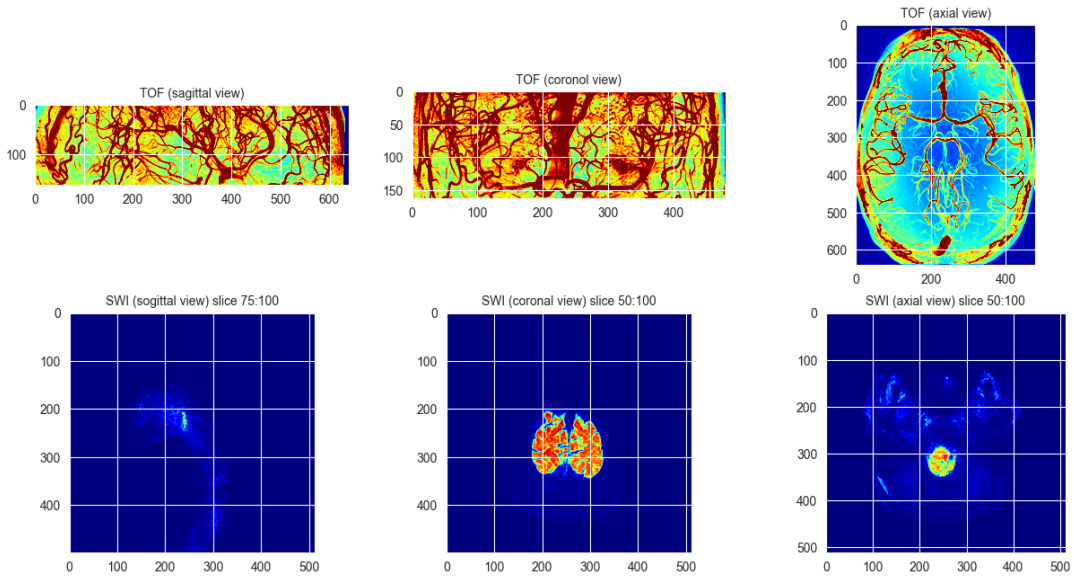
part1-a)




```
import os
import numpy as np
import nibabel as nib
import matplotlib.pyplot as plt
import seaborn as sns

#Dataset of image path
image_path_list = ['d:\ImgDrButtler\SWI.nii.gz', 'd:\ImgDrButtler\\TOF.nii.gz',
'd:\ImgDrButtler\T1.nii.gz', 'd:\ImgDrButtler\T2.nii.gz',
 'd:\ImgDrButtler\DWI.nii.gz', 'd:\ImgDrButtler\BOLD.nii.gz']

#Read image as nifti image
swi=nib.load(image_path_list[0]).get_fdata()
tof=nib.load(image_path_list[1]).get_fdata()
t1=nib.load(image_path_list[2]).get_fdata()
t2=nib.load(image_path_list[3]).get_fdata()
dwi=nib.load(image_path_list[4]).get_fdata()
bold=nib.load(image_path_list[5]).get_fdata()

#set style of backgroung
sns.set_style('darkgrid')


#function for show dimesion of images /show 2axes in z slice
#image(depth/2) / show 3D and 4D image by try except/plot setting
def show_3D_image(image_obj,titleP):
  image_data = image_obj.get_fdata()
  try:
    height, width, depth = image_data.shape
    half_depth = int(depth / 2)
```

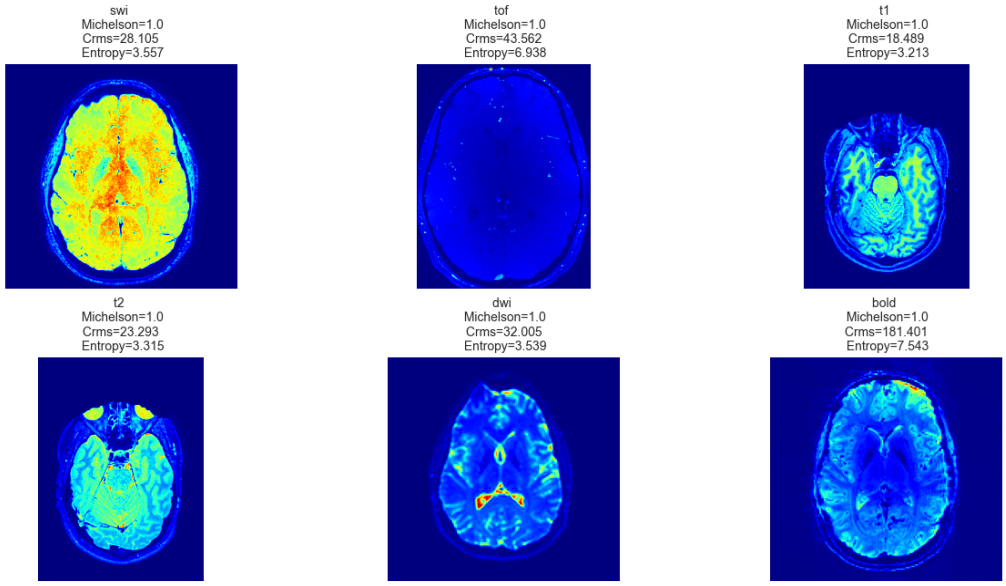
```
    plt.imshow(np.rot90(image_data[:, :, int(half_depth)]), cmap='jet')
  except:
     print('except')
     height, width, depth, channel = image_data.shape
     half_depth = int(depth / 2)
     plt.imshow(np.rot90(image_data[:, :, int(half_depth), 0]), cmap='jet')
  plt.title(titleP, fontsize=8)
  plt.axis('off')

num = 1
#create canvas for images
plt.figure()
print(len(image_path_list))

#for_loop for switch between images
for i in (image_path_list):
     #Read .nii file
    image_obj = nib.load(i)
    print(i)
    #devided plot for show 15 images-6 image for part1
    plt.subplot(2, 3, num)
    #call manually function by nibabel image and name of image
    show_3D_image(image_obj,os.path.basename(i))
    print(f'num={num}')
    num += 1

#finall show
plt.show()
```

Part 1-b)



```
#show colorful(jet) Min and Max Intensity(np.max|np.min) of TOF and SWI for each
dimension(axis=0|1|2)
#calculate np.max for tof image
plt.subplot(2,3,1)
plt.imshow(np.rot90(np.max(tof, axis=0)), cmap='jet', vmax=300)
plt.title('TOF (sagittal view)', fontsize=10)


plt.subplot(2,3,2)
plt.imshow(np.rot90(np.max(tof, axis=1)), cmap='jet', vmax=300)
plt.title('TOF (coronol view)', fontsize=10)

plt.subplot(2,3,3)
plt.imshow(np.rot90(np.max(tof, axis=2)), cmap='jet', vmax=300)
plt.title('TOF (axial view)', fontsize=10)

#calulate np.min for swi image
plt.subplot(2,3,4)
plt.imshow(np.rot90(np.min(swi[75:100,:,:], axis=0)), cmap='jet')
plt.title('SWI (sogittal view) slice 75:100', fontsize=10)


plt.subplot(2,3,5)
plt.imshow(np.rot90(np.min(swi[:, 50:100, :], axis=1)), cmap='jet')
plt.title('SWI (coronal view) slice 50:100', fontsize=10)


plt.subplot(2,3,6)
plt.imshow(np.rot90(np.min(nib.load(image_path_list[0]).get_fdata()[:,:,50:100],
axis=2)), cmap='jet')
plt.title('SWI (axial view) slice 50:100', fontsize=10)

plt.show()
```

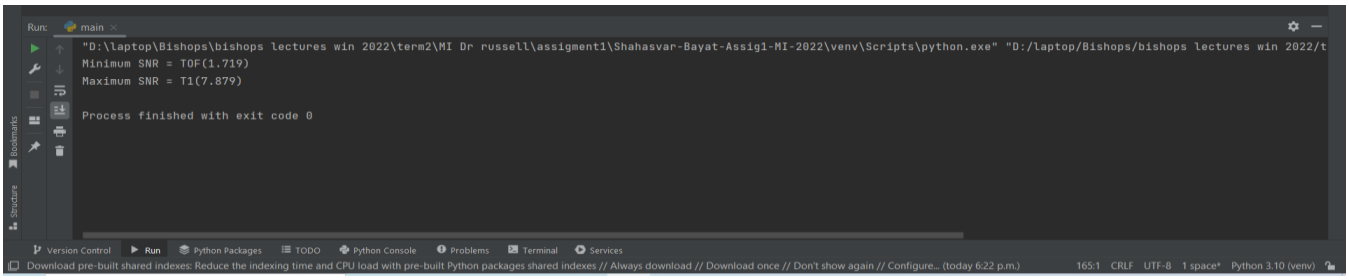
Part 2)




```
# Calculate Contrast of all images by Michelson - RMS -Entropy
import skimage.measure as msr
#calculate michelson contrast for each image
swi_Mc=((swi.max()-swi.min())/(swi.max()+swi.min()))
#calculate Rms contrast for each image
swi_Rms=np.round(np.sqrt(1/(swi.size-1)*np.sum((swi-np.mean(swi))**2)), 3)
#calculate Entropy for each image
swi_entr = np.round(msr.shannon_entropy(swi),3)

tof_Mc=((tof.max()-tof.min())/(tof.max()+tof.min()))
tof_Rms=np.round(np.sqrt(1/(tof.size-1)*np.sum((tof-np.mean(tof))**2)),3)
tof_entr = np.round(msr.shannon_entropy(tof),3)


t1_Mc=((t1.max()-t1.min())/(t1.max()+t1.min()))
t1_Rms=np.round(np.sqrt(1/(t1.size-1)*np.sum((t1-np.mean(t1))**2)),3)
t1_entr = np.round(msr.shannon_entropy(t1),3)

t2_Mc=((t2.max()-t2.min())/(t2.max()+t2.min()))
t2_Rms=np.round(np.sqrt(1/(t2.size-1)*np.sum((t2-np.mean(t2))**2)),3)
t2_entr = np.round(msr.shannon_entropy(t2),3)

dwi_Mc=((dwi.max()-dwi.min())/(dwi.max()+dwi.min()))
dwi_Rms=np.round(np.sqrt(1/(dwi.size-1)*np.sum((dwi-np.mean(dwi))**2)),3)
dwi_entr = np.round(msr.shannon_entropy(dwi),3)

bold_Mc=((bold.max()-bold.min())/(bold.max()+bold.min()))
bold_Rms=np.round(np.sqrt(1/(bold.size-1)*np.sum((bold-np.mean(bold))**2)),3)
bold_entr = np.round(msr.shannon_entropy(bold),3)

#plot all images with their contrast
```

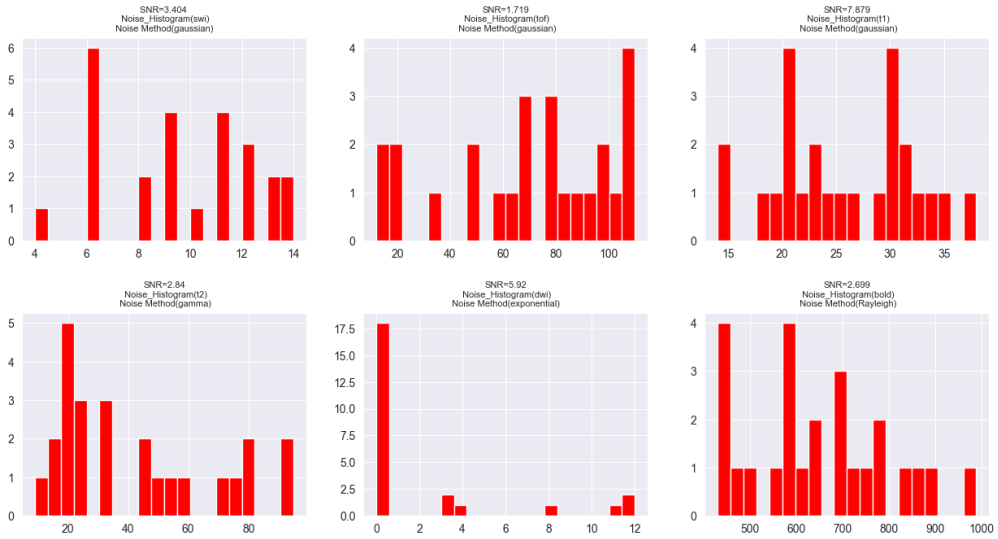
```
plt.subplot(2,3,1)
plt.imshow(np.rot90(swi[:,:,int(swi.shape[2]/2)]),cmap='jet')
plt.title(f'swi \n Michelson={swi_Mc}\n Crms={swi_Rms} \n
Entropy={swi_entr}',fontsize=10)
plt.axis('off')

plt.subplot(2,3,2)
plt.imshow(np.rot90(tof[:,:,int(tof.shape[2]/2)]),cmap='jet')
plt.title(f'tof \n Michelson={tof_Mc}\n Crms={tof_Rms} \n
Entropy={tof_entr}',fontsize=10)
plt.axis('off')

plt.subplot(2,3,3)
plt.imshow(np.rot90(t1[:,:,int(t1.shape[2]/2)]),cmap='jet')
plt.title(f't1 \n Michelson={t1_Mc}\n Crms={t1_Rms} \n
Entropy={t1_entr}',fontsize=10)
plt.axis('off')

plt.subplot(2,3,4)
plt.imshow(np.rot90(t2[:,:,int(t2.shape[2]/2)]),cmap='jet')
plt.title(f't2 \n Michelson={t2_Mc}\n Crms={t2_Rms} \n
Entropy={t2_entr}',fontsize=10)
plt.axis('off')

plt.subplot(2,3,5)
plt.imshow(np.rot90(dwi[:,:,int(dwi.shape[2]/2),0]),cmap='jet')
plt.title(f'dwi \n Michelson={dwi_Mc}\n Crms={dwi_Rms} \n
Entropy={dwi_entr}',fontsize=10)
plt.axis('off')
# print(dwi.shape)

plt.subplot(2,3,6)
plt.imshow(np.rot90(bold[:,:,int(bold.shape[2]/2),0]),cmap='jet')
plt.title(f'bold \n Michelson={bold_Mc}\n Crms={bold_Rms} \n
Entropy={bold_entr}',fontsize=10)
plt.axis('off')
```

Part 3-a)

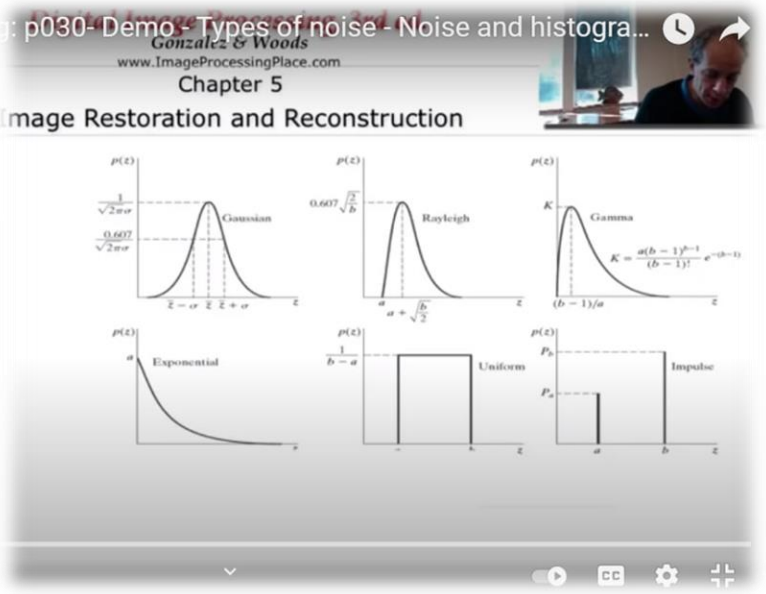



```
#Calculate SNR (Signal Noise Ratio(np.mean(main image)/np.std(noise))) for all
images
# plt.imshow(t2[:,:,200],cmap='jet',vmax=50)
SNR_swi=np.round(np.mean(swi[120:125,200:205,1])/np.std(swi[85:90,234:239,1]),3)
SNR_tof=np.round(np.mean(tof[156:161,326:331,1])/np.std(tof[10:15,362:367,1]),3)
SNR_t1=np.round(np.mean(t1[125:130,126:131,200])/np.std(t1[69:74,227:232,200]),3)
SNR_t2=np.round(np.mean(t2[104:109,208:213,200])/np.std(t2[101:106,235:240,200]),3)
SNR_dwi=np.round(np.mean(dwi[90:95,100:105,0,0])/np.std(dwi[95:100,109:114,0,0]),3)
SNR_bold=np.round(np.mean(bold[67:72,124:129,20,0])/np.std(bold[89:94,133:138,20,0]
),3)

#find and show the minimum SNR and maximum SNR
SNR_list=[SNR_swi,SNR_tof,SNR_t1,SNR_t2,SNR_dwi,SNR_bold]
SNR_list_name=["SWI","TOF","T1","T2","DWI","BOLD"]
print(f'Minimum SNR =
{SNR_list_name[SNR_list.index(np.min(SNR_list))]}({np.min(SNR_list)})')
print(f'Maximum SNR =
{SNR_list_name[SNR_list.index(np.max(SNR_list))]}({np.max(SNR_list)})')
plt.show()
```

Part 3-b)



According to the following image, we detect the noise method of each image by their histogram:

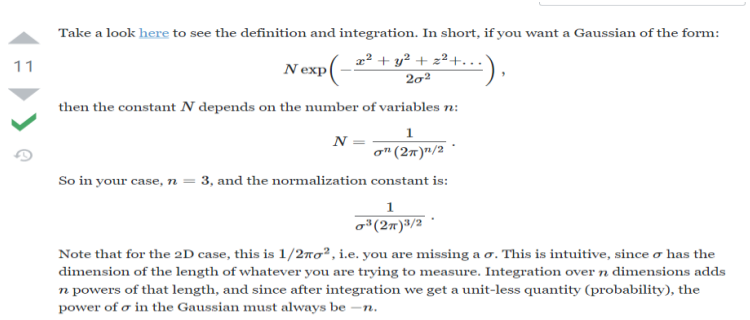



```
# Question Part  3-b-------------------------------------------------------------------
-------------
#plot Histogram of noise of all image with their SNRs and names
SNR_swi=np.round(np.mean(swi[120:125,200:205,1])/np.std(swi[85:90,234:239,1]),3)
SNR_tof=np.round(np.mean(tof[156:161,326:331,1])/np.std(tof[10:15,362:367,1]),3)
SNR_t1=np.round(np.mean(t1[125:130,126:131,200])/np.std(t1[69:74,227:232,200]),3)
SNR_t2=np.round(np.mean(t2[104:109,208:213,200])/np.std(t2[101:106,235:240,200]),3)
SNR_dwi=np.round(np.mean(dwi[90:95,100:105,0,0])/np.std(dwi[95:100,109:114,0,0]),3)
SNR_bold=np.round(np.mean(bold[67:72,124:129,20,0])/np.std(bold[89:94,133:138,20,0]
),3)

plt.subplot(2,3,1)
plt.hist(swi[85:90,234:239,1].ravel(),20,color='red')
plt.title(f'SNR={SNR_swi} \n Noise_Histogram(swi)\n Noise Method(gaussian)
',fontsize=8)

plt.subplot(2,3,2)
plt.hist(tof[10:15,362:367,1].ravel(),20,color='red')
plt.title(f'SNR={SNR_tof}\n Noise_Histogram(tof)\n Noise
Method(gaussian)',fontsize=8)

plt.subplot(2,3,3)
plt.hist(t1[69:74,227:232,200].ravel(),20,color='red')
plt.title(f'SNR={SNR_t1}\n Noise_Histogram(t1)\n Noise
Method(gaussian)',fontsize=8)

plt.subplot(2,3,4)
plt.hist(t2[101:106,235:240,200].ravel(),20,color='red')
plt.title(f'SNR={SNR_t2}\n Noise_Histogram(t2)\n Noise Method(gamma)',fontsize=8)

plt.subplot(2,3,5)
plt.hist(dwi[95:100,109:114,0,0].ravel(),20,color='red')
plt.title(f'SNR={SNR_dwi}\n Noise_Histogram(dwi)\n Noise Method(exponential)
```

```
',fontsize=8)

plt.subplot(2,3,6)
plt.hist(bold[89:94,133:138,20,0].ravel(),20,color='red')
plt.title(f'SNR={SNR_bold}\n Noise_Histogram(bold)\n Noise
Method(Rayleigh)',fontsize=8)
```

Part 4-a)

Calculate gaussian filter for noise reduction by bellow formula and then multiplying by frequency space representation of image(calculate furrier transform)



Take a look here to see the definition and integration. In short, if you want a Gaussian of the form:

$$N \exp\left(-\frac{x^2+y^2+z^2+\dots}{2\sigma^2}\right),$$

then the constant N depends on the number of variables n:

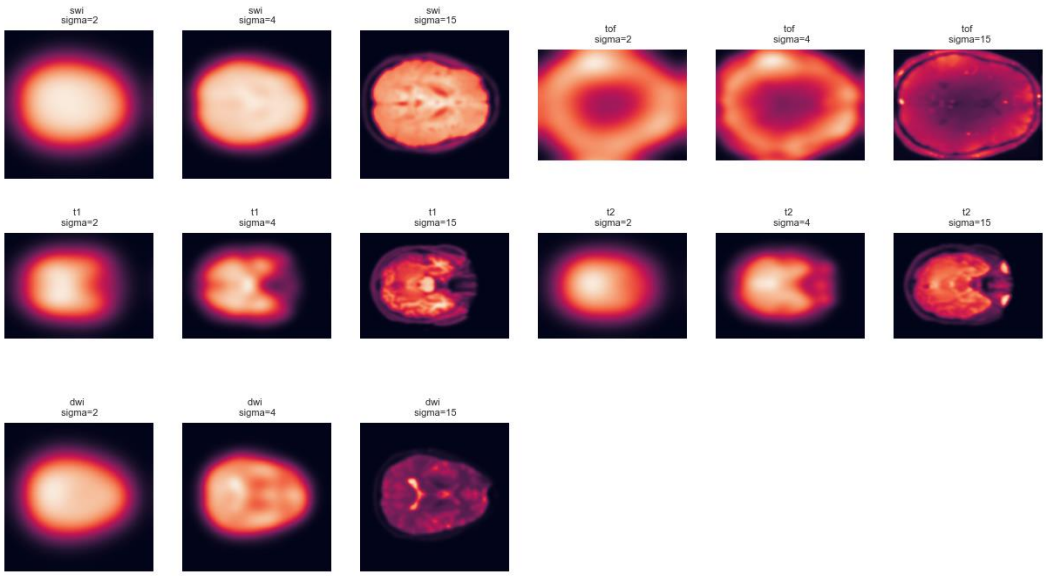
$$N = \frac{1}{\sigma^n (2\pi)^{n/2}}.$$

So in your case, $n = 3$, and the normalization constant is:

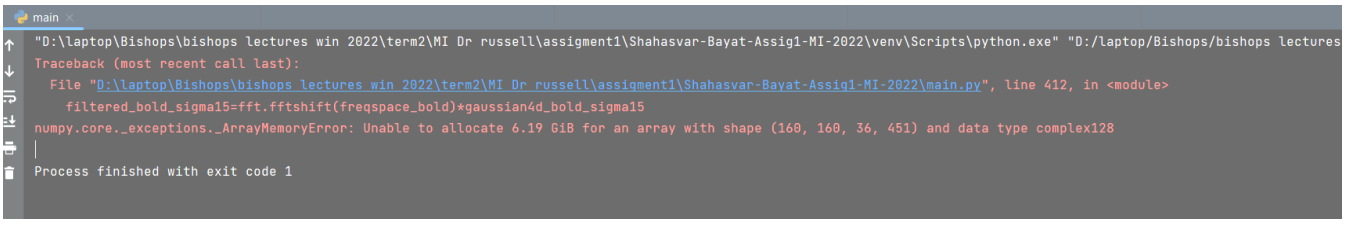
$$\frac{1}{\sigma^3 (2\pi)^{3/2}}.$$

Note that for the 2D case, this is $1/2\pi\sigma^2$, i.e. you are missing a σ. This is intuitive, since σ has the dimension of the length of whatever you are trying to measure. Integration over n dimensions adds n powers of that length, and since after integration we get a unit-less quantity (probability), the power of σ in the Gaussian must always be $-n$.

Due to the power of our computer, we had to run and capture part 4-a separately:



Notic: for 'bold' image,our system and also google colab couldn't show the output(because of low capacity of RAM). we show the error below(but we are sure that our code like for dwi image is correct) :



Code for 4-a)

```
#Question 4-a-------------------------------------------------
#we use linear filtering(gaussian) for noise reduction,
#that in this assigment we should multiplying in the
```

```
#frequency domain that we gain by furrior transform
import  numpy.fft as fft

#3 number of sigma
sigma_list=[2,4,15]

#swi.....................................................
#create frequency space by 3d furrior transform  for each image that has equal
shape with image
freqspace_swi=fft.fftn(swi)
#create 3d grid to define qaussian filter over it
xv,yv,zv=np.mgrid[-swi.shape[0]//2:swi.shape[0]//2,
                  -swi.shape[1]//2:swi.shape[1]//2,
                  -swi.shape[2]//2:swi.shape[2]//2]
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low
frequency
gaussian3d_swi_sigma2=(1/(sigma_list[0]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[0]**2))
gaussian3d_swi_sigma4=(1/(sigma_list[1]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[1]**2))
gaussian3d_swi_sigma15=(1/(sigma_list[2]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[2]**2))
#apply final step of filter on the image by multiplying qaussian function to
frequency space
filtered_swi_sigma2=fft.fftshift(freqspace_swi)*gaussian3d_swi_sigma2
filtered_swi_sigma4=fft.fftshift(freqspace_swi)*gaussian3d_swi_sigma4
filtered_swi_sigma15=fft.fftshift(freqspace_swi)*gaussian3d_swi_sigma15
#invert the image to show the image by ifftn
inv_swi_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered_swi_sigma2)))
inv_swi_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered_swi_sigma4)))
inv_swi_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered_swi_sigma15)))
#show the noise reduction of image
plt.subplot(3,6,1)
plt.imshow(inv_swi_sigma2[:,:,swi.shape[2]//2])
plt.title(f'swi \n sigma=2', fontsize=8)
plt.axis('off')
plt.subplot(3,6,2)
plt.imshow(inv_swi_sigma4[:,:,swi.shape[2]//2])
plt.title(f'swi \n sigma=4', fontsize=8)
plt.axis('off')
plt.subplot(3,6,3)
plt.imshow(inv_swi_sigma15[:,:,swi.shape[2]//2])
plt.title(f'swi \n sigma=15', fontsize=8)
plt.axis('off')

#Tof.....................................
#create frequency space by 3d furrior transform  for each image that has equal
shape with image
freqspace_tof=fft.fftn(tof)
#create 3d grid to define qaussian filter over it
xv,yv,zv=np.mgrid[-tof.shape[0]//2:tof.shape[0]//2,
                  -tof.shape[1]//2:tof.shape[1]//2,
                  -tof.shape[2]//2:tof.shape[2]//2]
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low
frequency
gaussian3d_tof_sigma2=(1/(sigma_list[0]**3*(2*np.pi)**(3/2)))*np.exp(-
```

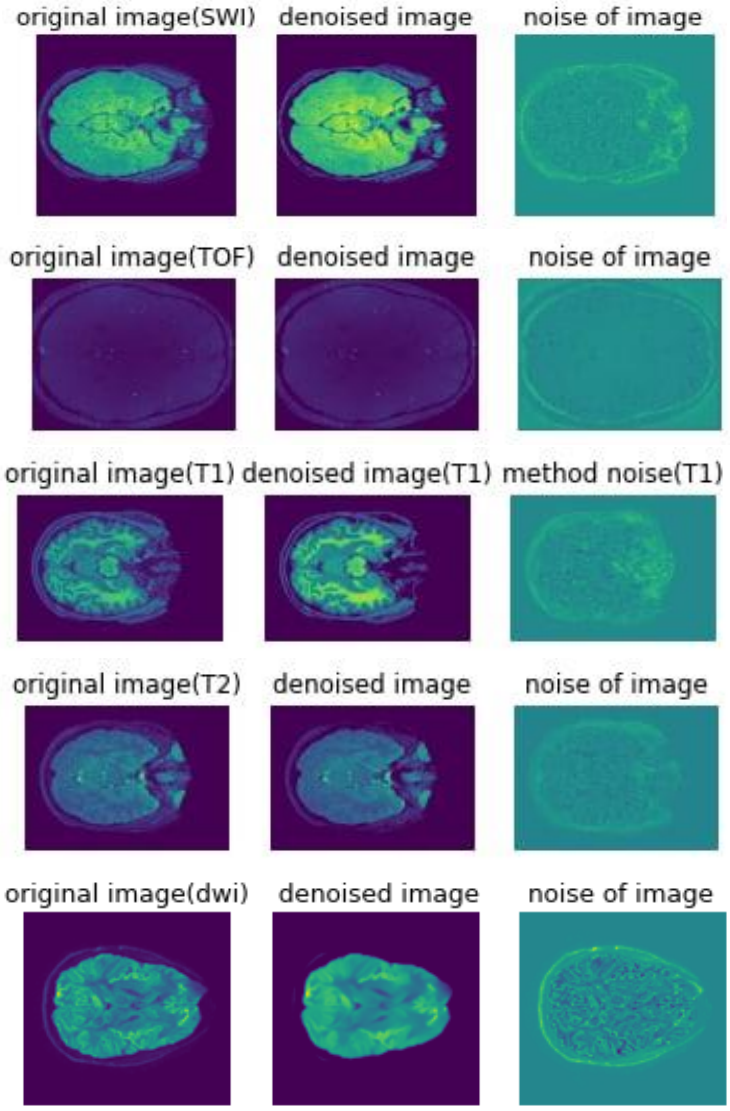
```
(xv**2+yv**2+zv**2)/(2*sigma_list[0]**2))
gaussian3d_tof_sigma4=(1/(sigma_list[1]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[1]**2))
gaussian3d_tof_sigma15=(1/(sigma_list[2]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[2]**2))
#apply final step of filter on the image by multiplying qaussian function to
frequency space
filtered_tof_sigma2=fft.fftshift(freqspace_tof)*gaussian3d_tof_sigma2
filtered_tof_sigma4=fft.fftshift(freqspace_tof)*gaussian3d_tof_sigma4
filtered_tof_sigma15=fft.fftshift(freqspace_tof)*gaussian3d_tof_sigma15
#invert the image to show the image by ifftn
inv_tof_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered_tof_sigma2)))
inv_tof_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered_tof_sigma4)))
inv_tof_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered_tof_sigma15)))
#show the noise reduction of image
plt.subplot(3,6,4)
plt.imshow(inv_tof_sigma2[:,:,tof.shape[2]//2])
plt.title(f'tof \n sigma=2', fontsize=8)
plt.axis('off')
plt.subplot(3,6,5)
plt.imshow(inv_tof_sigma4[:,:,tof.shape[2]//2])
plt.title(f'tof \n sigma=4', fontsize=8)
plt.axis('off')
plt.subplot(3,6,6)
plt.imshow(inv_tof_sigma15[:,:,tof.shape[2]//2])
plt.title(f'tof \n sigma=15', fontsize=8)
plt.axis('off')

#t1.....................................................
#create frequency space by 3d furrior transform  for each image that has equal
shape with image
freqspace_t1=fft.fftn(t1)
#create 3d grid to define qaussian filter over it
xv,yv,zv=np.mgrid[-t1.shape[0]//2:t1.shape[0]//2,
                  -t1.shape[1]//2:t1.shape[1]//2,
                  -t1.shape[2]//2:t1.shape[2]//2]
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low
frequency
gaussian3d_t1_sigma2=(1/(sigma_list[0]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[0]**2))
gaussian3d_t1_sigma4=(1/(sigma_list[1]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[1]**2))
gaussian3d_t1_sigma15=(1/(sigma_list[2]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[2]**2))
#apply final step of filter on the image by multiplying qaussian function to
frequency space
filtered_t1_sigma2=fft.fftshift(freqspace_t1)*gaussian3d_t1_sigma2
filtered_t1_sigma4=fft.fftshift(freqspace_t1)*gaussian3d_t1_sigma4
filtered_t1_sigma15=fft.fftshift(freqspace_t1)*gaussian3d_t1_sigma15
#invert the image to show the image by ifftn
inv_t1_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered_t1_sigma2)))
inv_t1_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered_t1_sigma4)))
inv_t1_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered_t1_sigma15)))
#show the noise reduction of image
plt.subplot(3,6,7)
plt.imshow(inv_t1_sigma2[:,:,t1.shape[2]//2])
```

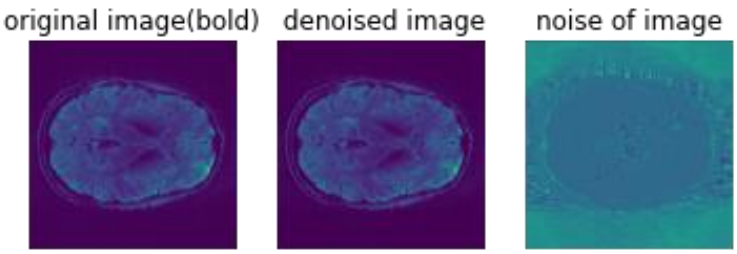
```
plt.title(f't1 \n sigma=2', fontsize=8)
plt.axis('off')
plt.subplot(3,6,8)
plt.imshow(inv_t1_sigma4[:,:,t1.shape[2]//2])
plt.title(f't1 \n sigma=4', fontsize=8)
plt.axis('off')
plt.subplot(3,6,9)
plt.imshow(inv_t1_sigma15[:,:,t1.shape[2]//2])
plt.title(f't1 \n sigma=15', fontsize=8)
plt.axis('off')

#t2.....................................................
#create frequency space by 3d furrior transform  for each image that has equal
shape with image
freqspace_t2=fft.fftn(t2)
#create 3d grid to define qaussian filter over it
xv,yv,zv=np.mgrid[-t2.shape[0]//2:t2.shape[0]//2,
                  -t2.shape[1]//2:t2.shape[1]//2,
                  -t2.shape[2]//2:t2.shape[2]//2]
#define gaussian 3d function to aatenuate the high frequency anf accenuate the low
frequency
gaussian3d_t2_sigma2=(1/(sigma_list[0]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[0]**2))
gaussian3d_t2_sigma4=(1/(sigma_list[1]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[1]**2))
gaussian3d_t2_sigma15=(1/(sigma_list[2]**3*(2*np.pi)**(3/2)))*np.exp(-
(xv**2+yv**2+zv**2)/(2*sigma_list[2]**2))
#apply final step of filter on the image by multiplying qaussian function to
frequency space
filtered_t2_sigma2=fft.fftshift(freqspace_t2)*gaussian3d_t2_sigma2
filtered_t2_sigma4=fft.fftshift(freqspace_t2)*gaussian3d_t2_sigma4
filtered_t2_sigma15=fft.fftshift(freqspace_t2)*gaussian3d_t2_sigma15
#invert the image to show the image by ifftn
inv_t2_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered_t2_sigma2)))
inv_t2_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered_t2_sigma4)))
inv_t2_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered_t2_sigma15)))
#show the noise reduction of image
plt.subplot(3,6,10)
plt.imshow(inv_t2_sigma2[:,:,t2.shape[2]//2])
plt.title(f't2 \n sigma=2', fontsize=8)
plt.axis('off')
plt.subplot(3,6,11)
plt.imshow(inv_t2_sigma4[:,:,t2.shape[2]//2])
plt.title(f't2 \n sigma=4', fontsize=8)
plt.axis('off')
plt.subplot(3,6,12)
plt.imshow(inv_t2_sigma15[:,:,t2.shape[2]//2])
plt.title(f't2 \n sigma=15', fontsize=8)
plt.axis('off')

#dwi.....................................................
#create frequency space by 4d furrior transform  for each image that has equal
shape with image
freqspace_dwi=fft.fftn(dwi)
#create 3d grid to define qaussian filter over it
xv,yv,zv,vv=np.mgrid[-dwi.shape[0]//2:dwi.shape[0]//2,
```

```
                  -dwi.shape[1]//2:dwi.shape[1]//2,
                  -dwi.shape[2]//2:dwi.shape[2]//2,
                  -dwi.shape[3]//2:dwi.shape[3]//2]
#define gaussian 4d function to aatenuate the high frequency anf accenuate the low
frequency
gaussian4d_dwi_sigma2=(1/(sigma_list[0]**4*(2*np.pi)**(4/2)))*np.exp(-
(xv**2+yv**2+zv**2+vv**2)/(2*sigma_list[0]**2))
gaussian4d_dwi_sigma4=(1/(sigma_list[1]**4*(2*np.pi)**(4/2)))*np.exp(-
(xv**2+yv**2+zv**2+vv**2)/(2*sigma_list[1]**2))
gaussian4d_dwi_sigma15=(1/(sigma_list[2]**4*(2*np.pi)**(4/2)))*np.exp(-
(xv**2+yv**2+zv**2+vv**2)/(2*sigma_list[2]**2))
#apply final step of filter on the image by multiplying qaussian function to
frequency space
filtered_dwi_sigma2=fft.fftshift(freqspace_dwi)*gaussian4d_dwi_sigma2
filtered_dwi_sigma4=fft.fftshift(freqspace_dwi)*gaussian4d_dwi_sigma4
filtered_dwi_sigma15=fft.fftshift(freqspace_dwi)*gaussian4d_dwi_sigma15
#invert the image to show the image by ifftn
inv_dwi_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered_dwi_sigma2)))
inv_dwi_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered_dwi_sigma4)))
inv_dwi_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered_dwi_sigma15)))
#show the noise reduction of image
plt.subplot(3,6,13)
plt.imshow(inv_dwi_sigma2[:,:,dwi.shape[2]//2,0])
plt.title(f'dwi \n sigma=2', fontsize=8)
plt.axis('off')
plt.subplot(3,6,14)
plt.imshow(inv_dwi_sigma4[:,:,dwi.shape[2]//2,0])
plt.title(f'dwi \n sigma=4', fontsize=8)
plt.axis('off')
plt.subplot(3,6,15)
plt.imshow(inv_dwi_sigma15[:,:,dwi.shape[2]//2,0])
plt.title(f'dwi \n sigma=15', fontsize=8)
plt.axis('off')

#bold.....................................................
#create frequency space by 3d furrior transform  for each image that has equal
shape with image
freqspace_bold=fft.fftn(bold)
#create 4d grid to define qaussian filter over it
xv,yv,zv,vv=np.mgrid[-bold.shape[0]//2:bold.shape[0]//2,
                  -bold.shape[1]//2:bold.shape[1]//2,
                  -bold.shape[2]//2:bold.shape[2]//2,
                  -bold.shape[3]//2:bold.shape[3]//2]
#define gaussian 4d function to aatenuate the high frequency anf accenuate the low
frequency
gaussian4d_bold_sigma2=(1/(sigma_list[0]**4*(2*np.pi)**(4/2)))*np.exp(-
(xv**2+yv**2+zv**2+vv**2)/(2*sigma_list[0]**2))
gaussian4d_bold_sigma4=(1/(sigma_list[1]**4*(2*np.pi)**(4/2)))*np.exp(-
(xv**2+yv**2+zv**2+vv**2)/(2*sigma_list[1]**2))
gaussian4d_bold_sigma15=(1/(sigma_list[2]**4*(2*np.pi)**(4/2)))*np.exp(-
(xv**2+yv**2+zv**2+vv**2)/(2*sigma_list[2]**2))
#apply final step of filter on the image by multiplying qaussian function to
frequency space
filtered_bold_sigma2=fft.fftshift(freqspace_bold)*gaussian4d_bold_sigma2
filtered_bold_sigma4=fft.fftshift(freqspace_bold)*gaussian4d_bold_sigma4
filtered_bold_sigma15=fft.fftshift(freqspace_bold)*gaussian4d_bold_sigma15
```

```
#invert the image to show the image by ifftn
inv_bold_sigma2=np.abs(fft.ifftn(fft.fftshift(filtered_bold_sigma2)))
inv_bold_sigma4=np.abs(fft.ifftn(fft.fftshift(filtered_bold_sigma4)))
inv_bold_sigma15=np.abs(fft.ifftn(fft.fftshift(filtered_bold_sigma15)))
#show the noise reduction of image
plt.subplot(3,6,16)
plt.imshow(inv_bold_sigma2[:,:,bold.shape[2]//2,0])
plt.title(f'bold \n sigma=2', fontsize=8)
plt.axis('off')
plt.subplot(3,6,17)
plt.imshow(inv_bold_sigma4[:,:,bold.shape[2]//2,0])
plt.title(f'bold \n sigma=4', fontsize=8)
plt.axis('off')
plt.subplot(3,6,18)
plt.imshow(inv_bold_sigma15[:,:,bold.shape[2]//2,0])
plt.title(f'bold \n sigma=15', fontsize=8)
plt.axis('off')
```

Part 4-b)






```
# Question 4_b-------------------------------------------------------
import dipy.denoise.nlmeans as nlmeans

# swi.................................
#calculate the denoised image of original image
den_swi = nlmeans.nlmeans(swi,10)

#show the original image with noise
plt.subplot(1,3,1)
plt.imshow(swi[:,:,200])
plt.title("original image(swi)")
plt.axis("off")

#show the denoised image
plt.subplot(1,3,2)
plt.imshow(den_swi[:,:,200])
plt.title("denoised image(swi)")
plt.axis("off")

#calculate and show the noise of image  by subtracing the denoise image of original
image
plt.subplot(1,3,3)
plt.imshow(swi[:,:200]-den_swi[:,:,200])
plt.title("noise of image(swi)")
plt.axis("off")


# tof.................................
den_tof = nlmeans.nlmeans(tof,10)
plt.subplot(1,3,1)
plt.imshow(tof[:,:,200])
plt.title("original image(tof)")
plt.axis("off")

plt.subplot(1,3,2)
plt.imshow(den_tof[:,:,200])
plt.title("denoised image(tof)")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(tof[:,:200]-den_tof[:,:,200])
plt.title("noise of image(tof)")
plt.axis("off")
```

```
# t1.............................
den_t1 = nlmeans.nlmeans(t1,10)
plt.subplot(1,3,1)
plt.imshow(t1[:,:,200])
plt.title("original image(t1)")
plt.axis("off")

plt.subplot(1,3,2)
plt.imshow(den_t1[:,:,200])
plt.title("denoised image(t1)")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(t1[:,:200]-den_t1[:,:,200])
plt.title("noise of image(t1)")
plt.axis("off")

# t2.............................
den_t2 = nlmeans.nlmeans(t2,10)
plt.subplot(1,3,1)
plt.imshow(t2[:,:,200])
plt.title("original image(t2)")
plt.axis("off")

plt.subplot(1,3,2)
plt.imshow(den_t2[:,:,200])
plt.title("denoised image(t2)")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(t2[:,:200]-den_t2[:,:,200])
plt.title("noise of image(t2)")
plt.axis("off")

# dwi.............................
den_dwi = nlmeans.nlmeans(dwi,10)
plt.subplot(1,3,1)
plt.imshow(dwi[:,:,35,10])
plt.title("original image(dwi)")
plt.axis("off")

plt.subplot(1,3,2)
plt.imshow(den_dwi[:,:,35,10])
plt.title(" denoised image")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(dwi[:,:,35,10]-den_dwi[:,:,35,10])
plt.title("noise of image")
plt.axis("off")

# bold.............................
den_bold = nlmeans.nlmeans(bold,10)
plt.subplot(1,3,1)
plt.imshow(bold[:,:,16,10])
plt.title("original image(bold)")
```

```
plt.axis("off")

plt.subplot(1,3,2)
plt.imshow(den_bold[:,:,16,10])
plt.title("denoised image(bold)")
plt.axis("off")

plt.subplot(1,3,3)
plt.imshow(bold[:,:,16,10]-den_bold[:,:,16,10])
plt.title("noise of image(bold)")
plt.axis("off")
plt.show()
```