Atanda Abdullahi Adewale (MSc. Computer Vision)

MRI IMAGE ANALYSIS:

- K-space manipulation
- qMRI
- Shepp-Logan phantom

In [ ]:
```
import pydicom
import matplotlib.pyplot as plt
from scipy.fftpack import fft2, fftshift, ifft2
import numpy as np

from skimage.data import shepp_logan_phantom
from scipy.optimize import curve_fit
```

In [ ]:
```
class Mri:
    def __init__(self, dcm):
        self.dcm = dcm

    def load_dcm (self):
        # Load DICOM file
        ds = pydicom.dcmread(self.dcm)
        # Get image data
        dcm_img = ds.pixel_array
        # Get high bit value
        high_bit = ds.HighBit
        # Print high bit value
        print(f"High bit value: {high_bit}")

        return dcm_img, high_bit

    def add_whitebox(self, image, hBit ):
        img_shape = image.shape
        center_x = img_shape[0] // 2
        center_y = img_shape[1] // 2
        box_size = 50
        image[center_x-box_size//2:center_x+box_size//2, center_y-box_si

        return image

    def rotate90(self, image):
        img_rotated = np.rot90(image, k=1)
        return img_rotated

    def show_image1(self, image, title):
        # Display the image with the box
```

```
        plt.imshow(image, cmap='gray')
        plt.title(title)
        plt.axis('off')
        plt.show()

    def show_image2(self, img1, img2, title1, title2):
        # Display both images
        fig, ax = plt.subplots(1, 2, figsize=(10, 5))
        ax[0].imshow(img1)
        ax[0].set_title(title1)
        ax[0].axis('off')
        ax[1].imshow(img2)
        ax[1].set_title(title2)
        ax[1].axis('off')
        plt.show()

    #k_space Generator
    def k_spacer(self, image):
        self.k_image = image
        # Apply 2D Fourier transform and shift zero-frequency component
        k_space = fftshift(fft2(self.k_image))

        # Take log of absolute value of k-space data for visualization
        k_log = 20*np.log(np.abs(k_space))

        return k_space, k_log

    # Inverse k_space transform to image
    def inv_k(k_space):
        recovered_img = ifft2(fftshift(k_space))
        #recovered_img = recovered_img.astype(np.uint8)
        rec_img = np.abs(recovered_img)
        return rec_img

class Mri_data():
    def __init__(self, dcm_file):
        self.dataset = pydicom.dcmread(dcm_file)

    def extract_info(self):
        # Extract patient information
        patient_name = self.dataset.PatientName
        patient_id = self.dataset.PatientID
        '''patient_age = self.dataset.PatientAge'''
        patient_sex = self.dataset.PatientSex

        # Extract image acquisition information
        scan_date = self.dataset.AcquisitionDate
        scan_time = self.dataset.AcquisitionTime
        scanner_type = self.dataset.ManufacturerModelName
        magnetic_field_strength = self.dataset.MagneticFieldStrength
        repetition_time = self.dataset.RepetitionTime
        echo_time = self.dataset.EchoTime
        slice_thickness = self.dataset.SliceThickness
        image_data = self.dataset.pixel_array # Extract image data
        image_orientation = self.dataset.ImageOrientationPatient # Extra
        image_contrast = self.dataset.ImageType # Extract image contrast
        '''signal_to_noise_ratio = self.dataset.ImageComments # Extract
        metadata = self.dataset.dir() # Extract metadata

        # Print the extracted information
```

```
        print('Patient Name:', patient_name)
        print('Patient ID:', patient_id)
        '''print('Patient Age:', patient_age)'''
        print('Patient Sex:', patient_sex)
        print('Scan Date:', scan_date)
        print('Scan Time:', scan_time)
        print('Scanner Type:', scanner_type)
        print('Magnetic Field Strength:', magnetic_field_strength)
        print('Repetition Time:', repetition_time)
        print('Echo Time:', echo_time)
        print('Slice Thickness:', slice_thickness)
        print('Image Data:', image_data)
        print('Image Orientation:', image_orientation)
        print('Image Contrast:', image_contrast)
        '''print('Signal to Noise Ratio:', signal_to_noise_ratio)'''
        print('Metadata:', metadata)

        '''or
        return {'Patient Name': patient_name, 'Patient ID': patient_id,
                'Patient Sex': patient_sex, 'Scan Date': scan_date, 'Sca
                'Scanner Type': scanner_type, 'Magnetic Field Strength':
                'Repetition Time': repetition_time, 'Echo Time': echo_ti
                'Image Data': image_data, 'Image Orientation': image_ori
                'Image Contrast': image_contrast, 'Signal to Noise Ratio
                'Metadata': metadata}
        '''


class display_npy():
    def __init__(self, npy_file):
        self.npy_file = npy_file

    def show(self):
        error_data = np.load(self.npy_file)

        # Apply inverse Fourier transform to convert k-space data to ima
        inv_error = Mri.inv_k(error_data)

        return inv_error

class kspace_edit():
    def __init__(self, kspace):
        self.kspace = kspace

    def remove_row_1over2(self):
        k1 = self.kspace[0::2]
        log_k1 = 20*np.log(np.abs(k1))
        return k1, log_k1

    def set_row_1over2_0(self):
        k_space2 = self.kspace
        k_space2[1::2]= 0

        epsilon = 1e-8
        k_eps = k_space2 + epsilon
        log_keps = 20*np.log(np.abs(k_eps))
        return k_eps, log_keps

    def row_interpolation(self):
        '''
```

```
        Interpolation row
        '''
        interpolated = self.kspace
        Ishape,_= interpolated.shape

        for i in range(1, Ishape-1):
            interpolated[i]= (1/2) * ( interpolated[i-1] + interpolated[
        log_interpolated = 20*np.log(np.abs(interpolated))

        return interpolated, log_interpolated

    def remove_col_1over2(self):
        k4 = self.kspace[:,1::2]
        log_k4 = 20*np.log(np.abs(k4))
        return k4, log_k4

    def set_col_1over2_0(self):

        k5 = self.kspace
        k5[:,1::2] = 0

        epsilon2 = 1e-8
        k_eps2 = k5 + epsilon2
        log_keps2 = 20*np.log(np.abs(k_eps2))

        return k_eps2, log_keps2

    def col_interpolation(self):
        '''
        Interpolation col
        '''
        inter_c = self.kspace
        _, ic = inter_c.shape

        for i in range(1, ic-1):
            inter_c[:, i]= (1/2) * ( inter_c[:, i-1] + inter_c[:, i+1] )

        log_inter_c = 20*np.log(np.abs(inter_c))

        return inter_c, log_inter_c
```

Question 1.1 :

Open the only DICOM image available in the dataset with pydicom, and give me the important informations of it (use pydicom documentation).

In [ ]:
```
# Extract information from the DICOM file
extracted_info = Mri_data('t1.dcm')
data_extracted = extracted_info.extract_info()
```

```
Patient Name: t1
Patient ID: t1
Patient Sex: O
Scan Date: 20230426
Scan Time: 183221.731701
Scanner Type: Verio
Magnetic Field Strength: 3
Repetition Time: 1700
Echo Time: 2.83
Slice Thickness: 0.90000003576279
Image Data: [[0 0 0 ... 0 0 0]
 [0 1 1 ... 1 0 0]
 [0 1 0 ... 0 0 1]
 ...
 [0 1 1 ... 1 0 1]
 [0 0 1 ... 1 2 2]
 [0 0 0 ... 1 1 1]]
Image Orientation: [0.99992674591514, -0.0015877929155, -0.011999238193
3, 2.061543e-010, 0.99135843833204, -0.1311809694577]
Image Contrast: ['ORIGINAL', 'PRIMARY', 'M', 'ND', 'NORM', 'FM2_2', 'FI
L']
Metadata: ['AccessionNumber', 'AcquisitionDate', 'AcquisitionMatrix',
'AcquisitionNumber', 'AcquisitionTime', 'AngioFlag', 'BitsAllocated',
'BitsStored', 'Columns', 'ContentDate', 'ContentTime', 'Deidentificatio
nMethod', 'DeidentificationMethodCodeSequence', 'DeviceSerialNumber',
'EchoNumbers', 'EchoTime', 'EchoTrainLength', 'FlipAngle', 'FrameOfRefe
renceUID', 'HighBit', 'ImageOrientationPatient', 'ImagePositionPatien
t', 'ImageType', 'ImagedNucleus', 'ImagingFrequency', 'InPlanePhaseEnco
dingDirection', 'InstanceCreationDate', 'InstanceCreationTime', 'Instan
ceNumber', 'InstitutionName', 'InversionTime', 'LargestImagePixelValu
e', 'LossyImageCompression', 'MRAcquisitionType', 'MagneticFieldStrengt
h', 'Manufacturer', 'ManufacturerModelName', 'Modality', 'NumberOfAvera
ges', 'NumberOfPhaseEncodingSteps', 'OperatorsName', 'PatientBirthDat
e', 'PatientID', 'PatientIdentityRemoved', 'PatientName', 'PatientPosit
ion', 'PatientSex', 'PercentPhaseFieldOfView', 'PercentSampling', 'Phot
ometricInterpretation', 'PixelBandwidth', 'PixelData', 'PixelRepresenta
tion', 'PixelSpacing', 'PositionReferenceIndicator', 'ProcedureCodeSequ
ence', 'ProtocolName', 'ReferringPhysicianName', 'RepetitionTime', 'Req
uestedProcedureCodeSequence', 'RequestedProcedureDescription', 'Rows',
'SAR', 'SOPClassUID', 'SOPInstanceUID', 'SamplesPerPixel', 'ScanOption
s', 'ScanningSequence', 'SequenceName', 'SequenceVariant', 'SeriesDat
e', 'SeriesInstanceUID', 'SeriesNumber', 'SeriesTime', 'SliceLocation',
'SliceThickness', 'SmallestImagePixelValue', 'SoftwareVersions', 'Speci
ficCharacterSet', 'StationName', 'StudyDate', 'StudyID', 'StudyInstance
UID', 'StudyTime', 'TransmitCoilName', 'VariableFlipAngleFlag', 'Window
Center', 'WindowCenterWidthExplanation', 'WindowWidth', 'dBdt']
```

Above are informations from t1.dcm, a DICOM image of the brain computed from magnetic resonance imaging (MRI)

Question 1.2 :

Change the name, surname of the file and save it with the new name. Re-open it and ensure that the modifications were taken into account

In [ ]:
```
# Modify the patient name
extracted_info.dataset.PatientName = 'Adewale'
```

```
# Save the modified DICOM file with the new patient name
pydicom.dcmwrite('adewale.dcm', extracted_info.dataset)
```

In [ ]:
```
# Extract information from the DICOM file
extracted_info = Mri_data('adewale.dcm')
data_extracted = extracted_info.extract_info()
```

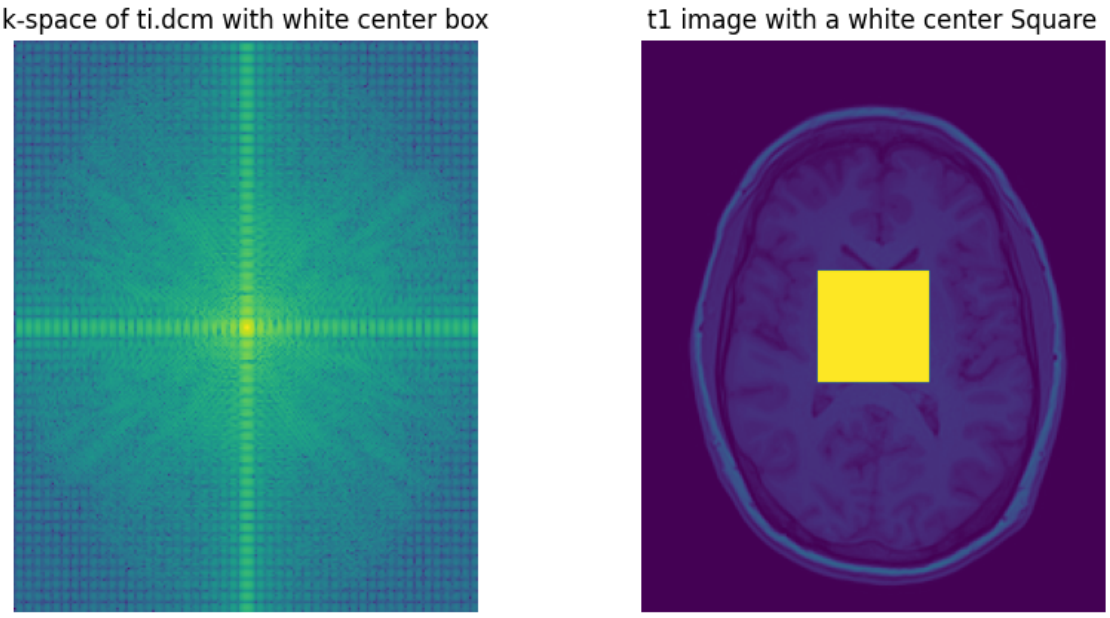
```
Patient Name: Adewale
Patient ID: t1
Patient Sex: O
Scan Date: 20230426
Scan Time: 183221.731701
Scanner Type: Verio
Magnetic Field Strength: 3
Repetition Time: 1700
Echo Time: 2.83
Slice Thickness: 0.90000003576279
Image Data: [[0 0 0 ... 0 0 0]
 [0 1 1 ... 1 0 0]
 [0 1 0 ... 0 0 1]
 ...
 [0 1 1 ... 1 0 1]
 [0 0 1 ... 1 2 2]
 [0 0 0 ... 1 1 1]]
Image Orientation: [0.99992674591514, -0.0015877929155, -0.011999238193
3, 2.061543e-010, 0.99135843833204, -0.1311809694577]
Image Contrast: ['ORIGINAL', 'PRIMARY', 'M', 'ND', 'NORM', 'FM2_2', 'FI
L']
Metadata: ['AccessionNumber', 'AcquisitionDate', 'AcquisitionMatrix',
'AcquisitionNumber', 'AcquisitionTime', 'AngioFlag', 'BitsAllocated',
'BitsStored', 'Columns', 'ContentDate', 'ContentTime', 'DeidentificatioMethod', 'DeidentificationMethodCodeSequence', 'DeviceSerialNumber',
'EchoNumbers', 'EchoTime', 'EchoTrainLength', 'FlipAngle', 'FrameOfRefeReferenceUID', 'HighBit', 'ImageOrientationPatient', 'ImagePositionPatient', 'ImageType', 'ImagedNucleus', 'ImagingFrequency', 'InPlanePhaseEncodingDirection', 'InstanceCreationDate', 'InstanceCreationTime', 'InstanceNumber', 'InstitutionName', 'InversionTime', 'LargestImagePixelValue', 'LossyImageCompression', 'MRAcquisitionType', 'MagneticFieldStrength', 'Manufacturer', 'ManufacturerModelName', 'Modality', 'NumberOfAverages', 'NumberOfPhaseEncodingSteps', 'OperatorsName', 'PatientBirthDate', 'PatientID', 'PatientIdentityRemoved', 'PatientName', 'PatientPosition', 'PatientSex', 'PercentPhaseFieldOfView', 'PercentSampling', 'PhotometricInterpretation', 'PixelBandwidth', 'PixelData', 'PixelRepresentation', 'PixelSpacing', 'PositionReferenceIndicator', 'ProcedureCodeSequence', 'ProtocolName', 'ReferringPhysicianName', 'RepetitionTime', 'RequestedProcedureCodeSequence', 'RequestedProcedureDescription', 'Rows',
'SAR', 'SOPClassUID', 'SOPInstanceUID', 'SamplesPerPixel', 'ScanOptions', 'ScanningSequence', 'SequenceName', 'SequenceVariant', 'SeriesDate', 'SeriesInstanceUID', 'SeriesNumber', 'SeriesTime', 'SliceLocation',
'SliceThickness', 'SmallestImagePixelValue', 'SoftwareVersions', 'SpecificCharacterSet', 'StationName', 'StudyDate', 'StudyID', 'StudyInstanceUID', 'StudyTime', 'TransmitCoilName', 'VariableFlipAngleFlag', 'WindowCenter', 'WindowCenterWidthExplanation', 'WindowWidth', 'dBdt']
```

Question 1.3 :

Add a white square within the image and save it in a new DICOM. Open this new image and ensure the white square presence in the center of the image.

In [ ]:
```
t1_box = Mri('t1.dcm')
t1_image, h_bit = t1_box.load_dcm()
t1_boxed = t1_box.add_whitebox(t1_image, h_bit)

tk, t1_box_kspace = t1_box.k_spacer(t1_boxed)

t1_boxed_title = "t1 image with a white center Square"
t1_box_kspace_title = "k-space of ti.dcm with white center box"

show_t1_boxed = t1_box.show_image2(t1_box_kspace, t1_boxed, t1_box_kspac


# Create new DICOM file with modified image
ds_new = pydicom.dcmread('t1.dcm')
ds_new.PixelData = t1_boxed.tobytes()
ds_new.save_as('t1_boxed.dcm')
```

```
High bit value: 11
```

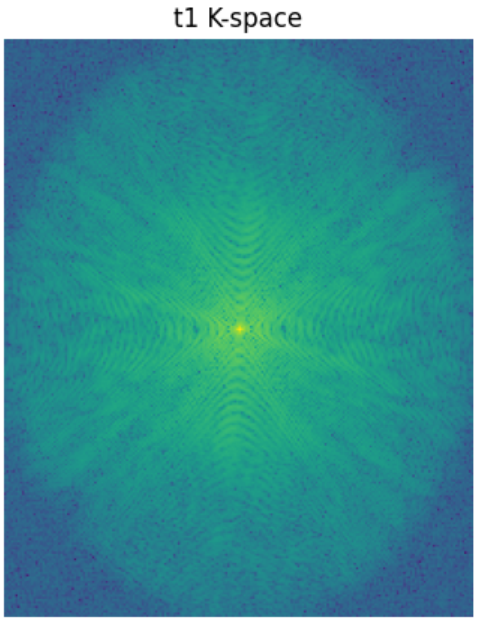


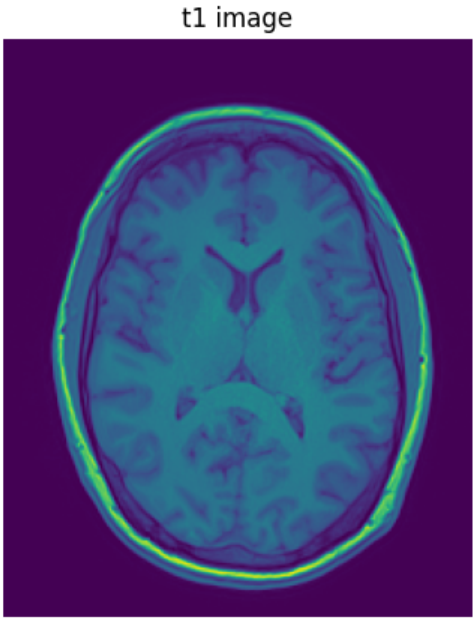
Question 2.1 :

The first task is to open the DICOM image (t1.dcm) and display it using the matplotlib library. Display also the k-Space. You can create a function, il will be used often.

In [ ]:
```
t1 = Mri('t1.dcm')
t1_img, high_bit = t1.load_dcm()
t1k,log_t1  = t1.k_spacer(t1_img)


title1 = 't1 K-space'
title2 = 't1 image'
show_t1 = t1.show_image2(log_t1, t1_img, title1, title2)
```

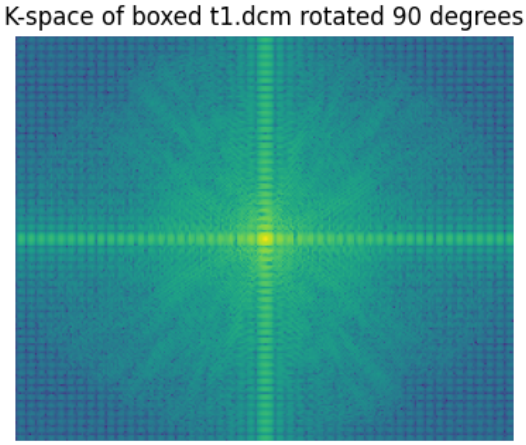
```
High bit value: 11
```

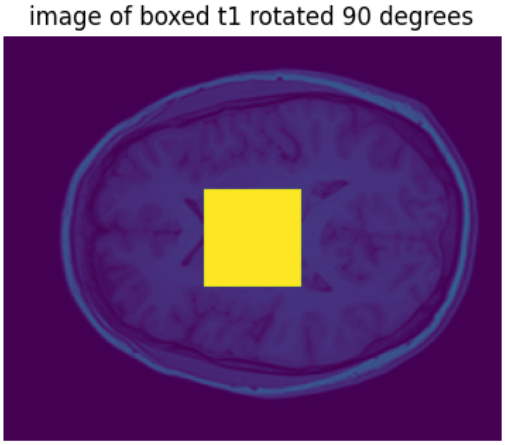




Question 2.2 :
The, rotate 90 degrees the image and observe the impact in the k-Space. What is the impact of the rotation ?

In [ ]:
```
t1_box_rot90 = t1_box.rotate90(t1_boxed)
tkb, t1_box_kspace_rot90 = t1_box.k_spacer(t1_box_rot90)

title3 = 'K-space of boxed t1.dcm rotated 90 degrees'
title4 = 'image of boxed t1 rotated 90 degrees'
show_t1 = t1.show_image2(t1_box_kspace_rot90, t1_box_rot90, title3, titl
```





A 90 degree rotation in k-space is equivalent to swapping the roles of the frequency and phase encoding directions.

In above MRI k-space, after a 90 degree rotation, there is a subtle image distortion and loss of image contrast.

Question 2.3 :
Transform file error.npy into the image space.

In [ ]:
```
show_npy = display_npy('error.npy')
show_error = show_npy.show()
```

```
print(show_error.shape)

error_title = "Error.npy Image"
error = t1_box.show_image1(show_error, error_title)
```

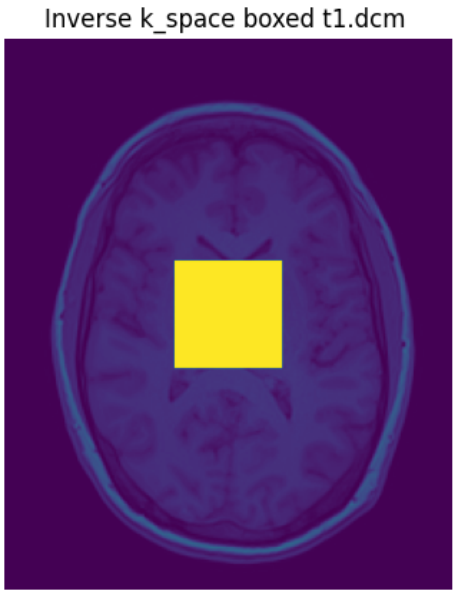
```
(550, 550)
```

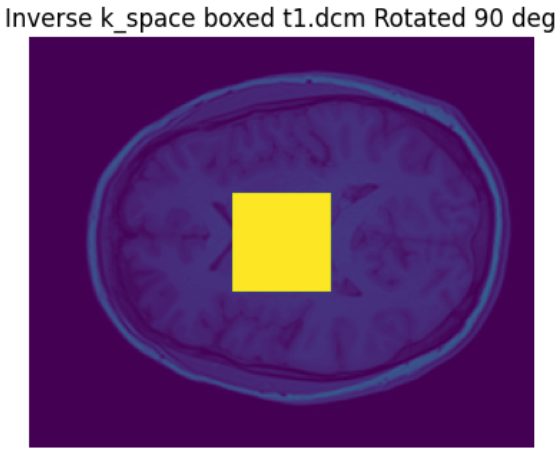
Generated QR code from error.npy is an hyperlink to Rickrolling or a Rickroll is an internet meme involving the unexpected appearance of the music video for the 1987 song "Never Gonna Give You Up", performed by English singer Rick Astley. The aforementioned video has over 1 billion views on YouTube.

Question 2.4 :
We propose two k-space treatments: removing the center and removing the outside

Describe the impact of each treatment in the k-Space on the final image.

In [ ]:
```
inv_t1boxed = Mri.inv_k(tk)
inv_t1boxed_rot90 = Mri.inv_k(tkb)

title5 = 'Inverse k_space boxed t1.dcm '
title6 = 'Inverse k_space boxed t1.dcm Rotated 90 deg'
show_t1k_boxed_rot = t1.show_image2(inv_t1boxed, inv_t1boxed_rot90, titl
```

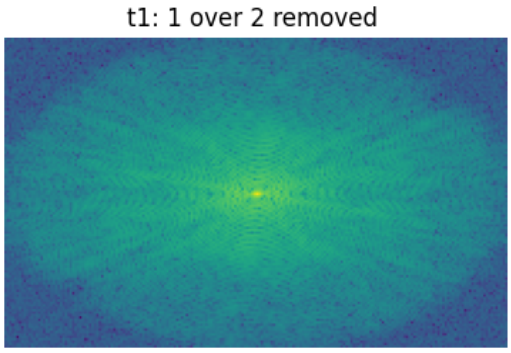




Question 3.1 :

First, remove one line over two and reconstruct the corresponding image.

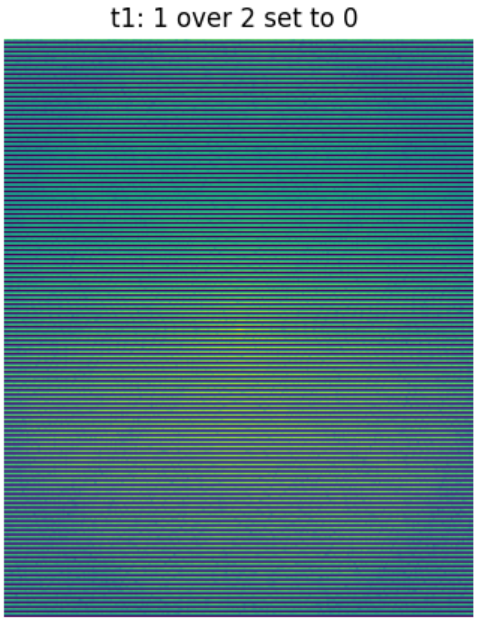
In [ ]:
```
kspace_editn = kspace_edit(t1k)

oneOverTwo, log_oneOverTwo = kspace_editn.remove_row_1over2()
inv_t1over2_img = Mri.inv_k(oneOverTwo)

title7 = 't1: 1 over 2 removed '
title8 = 'inverse t1: 1 over 2 removed'
show_t1over2 = t1.show_image2(log_oneOverTwo, inv_t1over2_img, title7, t
```

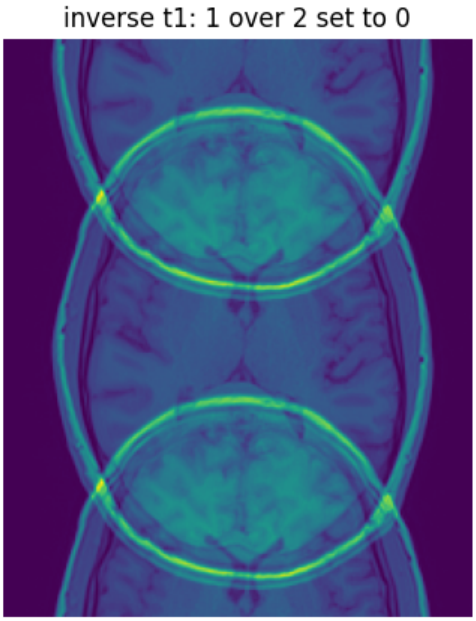


Question 3.2 :

Second, instead of removing lines you will fill by 0 these lines. Again, compute the resulting image.

In [ ]:
```
oneOverTwo_0, log_oneOverTwo_0 = kspace_editn.set_row_1over2_0()

inv_t1over2_0_img = Mri.inv_k(oneOverTwo_0)

title9 = 't1: 1 over 2 set to 0 '
title10 = 'inverse t1: 1 over 2 set to 0'
show_t1over2_0 = t1.show_image2(log_oneOverTwo_0, inv_t1over2_0_img, tit
```



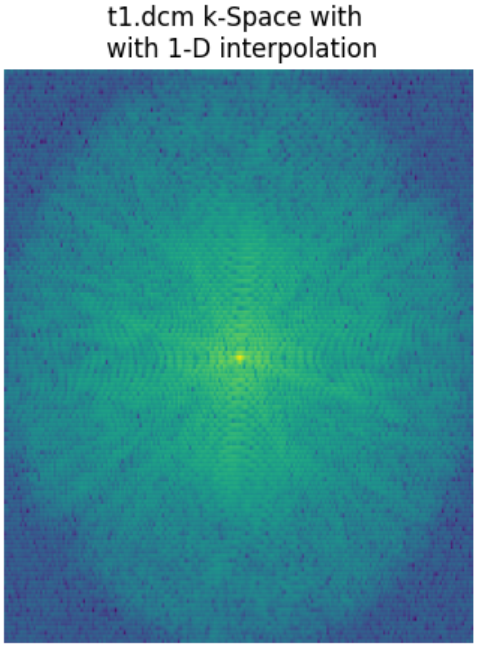


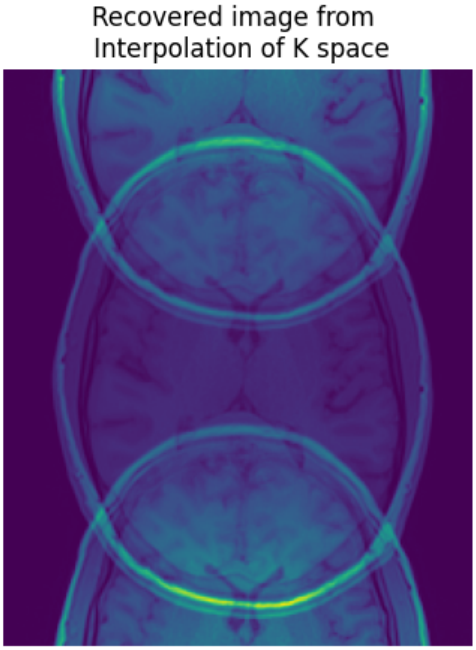
Question 3.3 :

Third, you will interpolate the lines filled by 0 to closest values in the k-Space. Compute the corresponding image and comment the result regarding the two previous exercices

In [ ]:
```
interpolation, log_interpolation = kspace_editn.row_interpolation()

inv_row_interpolate = Mri.inv_k(interpolation)

title11 = 't1.dcm k-Space with \n with 1-D interpolation'
title12 = 'Recovered image from \n Interpolation of K space'
show_t1_interpolate = t1.show_image2(log_interpolation, inv_row_interpol
```

```
/tmp/ipykernel_29776/195002975.py:159: RuntimeWarning: divide by zero e
ncountered in log
  log_interpolated = 20*np.log(np.abs(interpolated))
```





In [ ]:
```

```

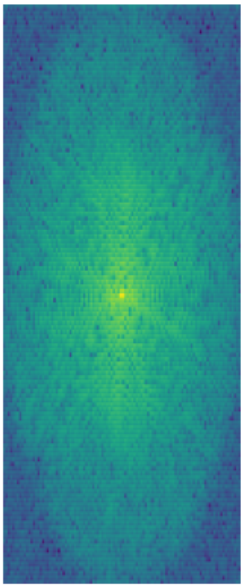
Question 3.4 :

Please repeat the entire exercise, but instead of applying the processing steps to each row, apply them to each column. Once complete, please provide commentary on the results and compare them to the results obtained from the row approach.

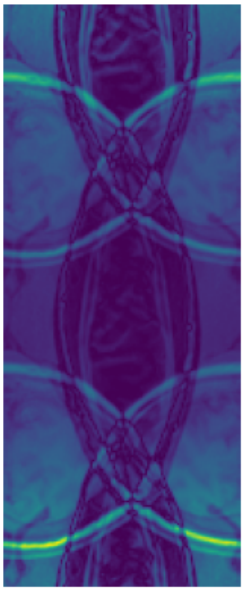
In [ ]:
```
colspace, log_colspace = kspace_editn.remove_col_1over2()
inv_col = Mri.inv_k(colspace)
title13 = 't1-kspace: 1over2 columns removed'
title14 = 'inverse t1: 1over2 columns removed'
show_t1_col = t1.show_image2(log_colspace, inv_col, title13, title14)


colspace_0, log_colspace_0 = kspace_editn.set_col_1over2_0()
inv_col_0 = Mri.inv_k(colspace_0)
title15 = 't1-kspace: 1over2 columns set to 0 '
title16 = 'inverse t1: 1over2 columns set to 0'
show_t1_col_o = t1.show_image2(log_colspace_0, inv_col_0, title15, title


col_inter, log_col_inter = kspace_editn.col_interpolation()
inv_col_interpolate = Mri.inv_k(col_inter)
title16 = 't1.dcm k-Space column 1-D interpolation'
title17 = 'Recovered image column Interpolation'
show_t1_col_interpolate = t1.show_image2(log_col_inter, inv_col_interpol
```

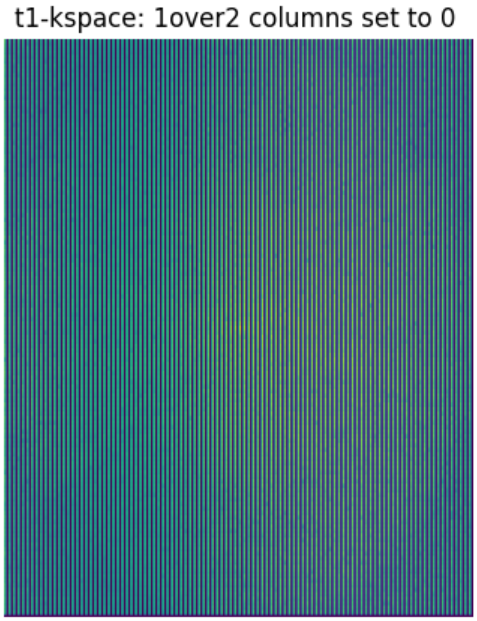
```
/tmp/ipykernel_29776/195002975.py:165: RuntimeWarning: divide by zero e
ncountered in log
  log_k4 = 20*np.log(np.abs(k4))
```

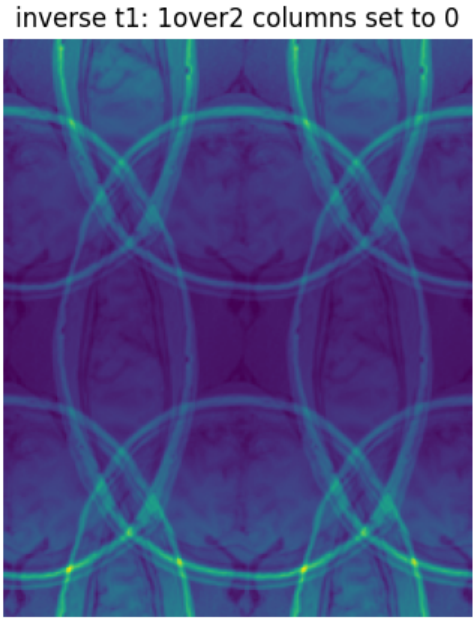
t1-kspace: 1over2 columns removed



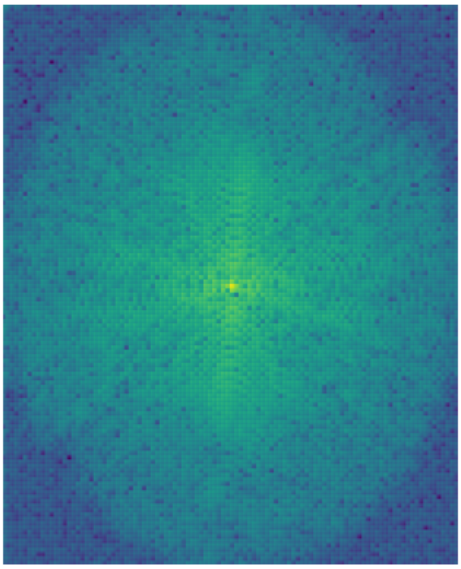
inverse t1: 1over2 columns removed

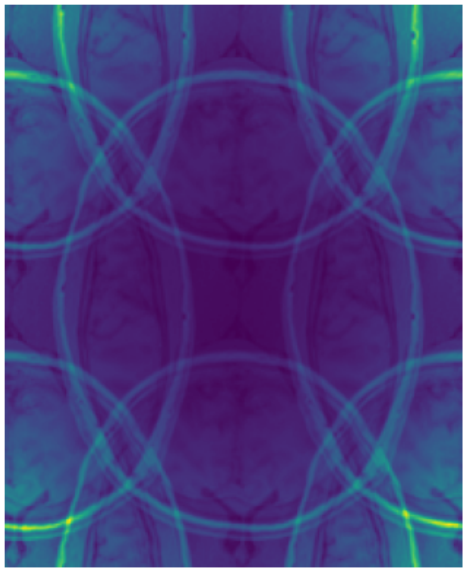







```
/tmp/ipykernel_29776/195002975.py:189: RuntimeWarning: divide by zero e
ncountered in log
  log_inter_c = 20*np.log(np.abs(inter_c))
```





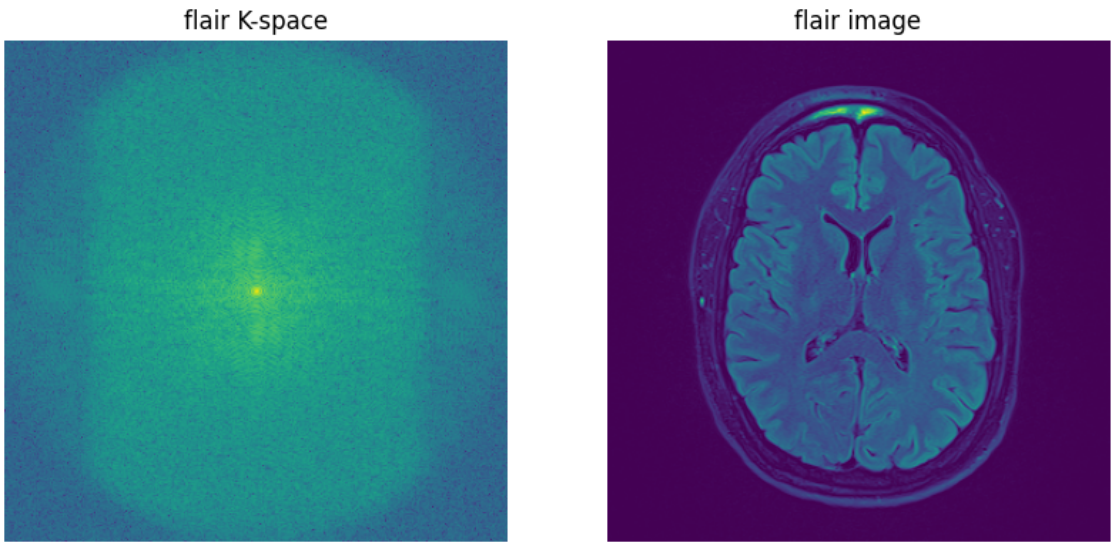
Question 3.5 :

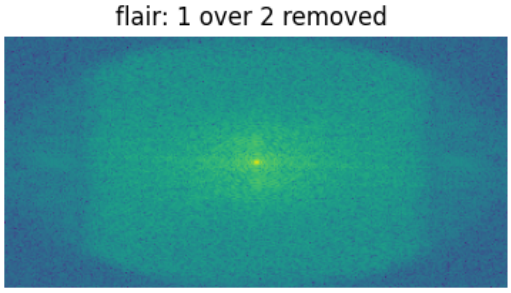
Assuming your code is functional, please replicate the same image processing steps using the file named flair.dcm

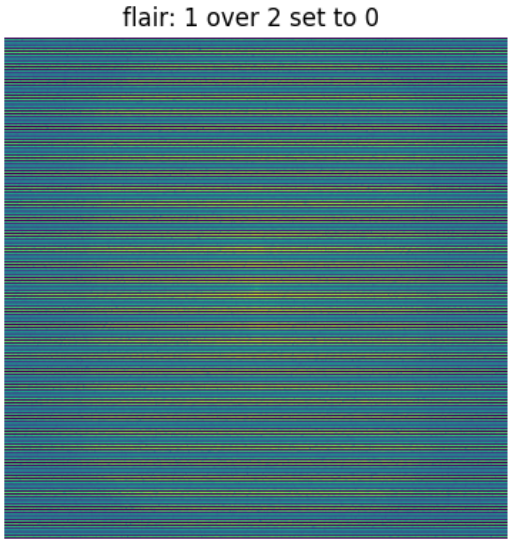
In [ ]:
```
flair = Mri('flair.dcm')

flair_img, high_bitf = flair.load_dcm()
flairk,log_flair  = flair.k_spacer(flair_img)
title_flair1 = 'flair K-space'
title_flair2 = 'flair image'
show_flair = flair.show_image2(log_flair, flair_img, title_flair1, title


flair_kspace= kspace_edit(flairk)
a, b = flair_kspace.remove_row_1over2()
inv_flair = Mri.inv_k(a)
title_f1 = 'flair: 1 over 2 removed '
```

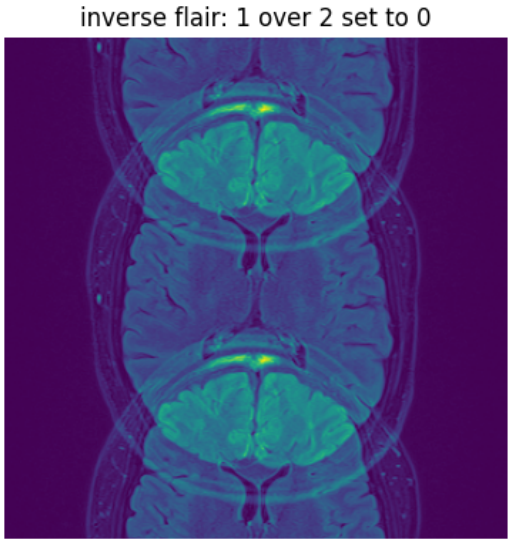
```
title_f2 = 'inverse flair: 1 over 2 removed'
show_flairover2 = flair.show_image2(b, inv_flair, title_f1, title_f2)


c, d = flair_kspace.set_row_1over2_0()
inv_flair_0 = Mri.inv_k(c)
title_f3 = 'flair: 1 over 2 set to 0 '
title_f4 = 'inverse flair: 1 over 2 set to 0'
show_flairover2_0 = flair.show_image2(d, inv_flair_0, title_f3, title_f4


e, f = flair_kspace.row_interpolation()
flair_inv_row_inter = Mri.inv_k(e)

title_f5 = 'flair.dcm k-Space with \n with 1-D interpolation'
title_f6 = 'Recovered image from \n Interpolation of flair.dcm K space'
show_flair_interpolate = flair.show_image2(f, flair_inv_row_inter, title

g, h = flair_kspace.remove_col_1over2()
inv_fcol = Mri.inv_k(g)
title_f7 = 'flair-kspace: 1over2 columns removed'
title_f8 = 'inverse flair: 1over2 columns removed'
show_flair_col = flair.show_image2(h, inv_fcol, title_f7, title_f8)


i, j = flair_kspace.set_col_1over2_0()
inv_fcol_0 = Mri.inv_k(i)
title_f9 = 'flair-kspace: 1over2 columns set to 0 '
title_f10 = 'inverse flair: 1over2 columns set to 0'
show_flair_col_o = flair.show_image2(j, inv_col_0, title_f9, title_f10)


k, l = flair_kspace.col_interpolation()
inv_fcol_interpolate = Mri.inv_k(k)
title_f11 = 'fliar.dcm k-Space column 1-D interpolation'
title_f12 = 'Recovered flair image column Interpolation'
show_flair_col_interpolate = flair.show_image2(l, inv_fcol_interpolate,
```

```
High bit value: 11
```

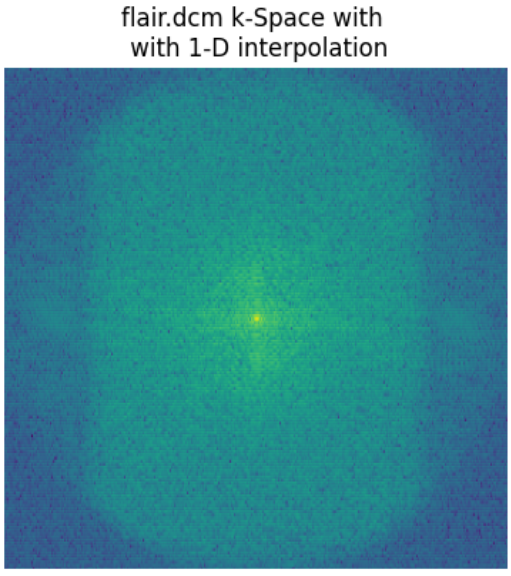


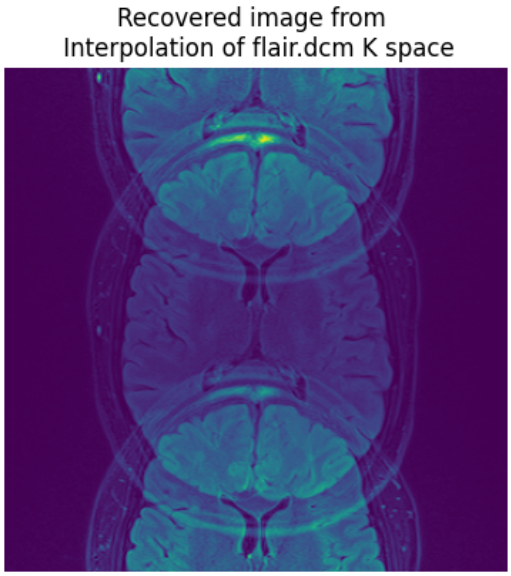






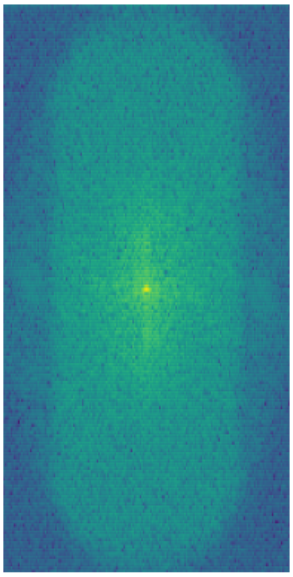

```
/tmp/ipykernel_29776/195002975.py:159: RuntimeWarning: divide by zero e
ncountered in log
  log_interpolated = 20*np.log(np.abs(interpolated))
```



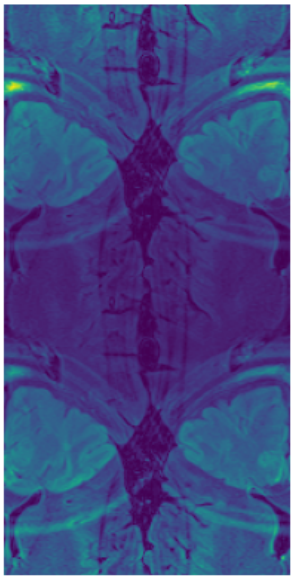



```
/tmp/ipykernel_29776/195002975.py:165: RuntimeWarning: divide by zero e
ncountered in log
  log_k4 = 20*np.log(np.abs(k4))
```

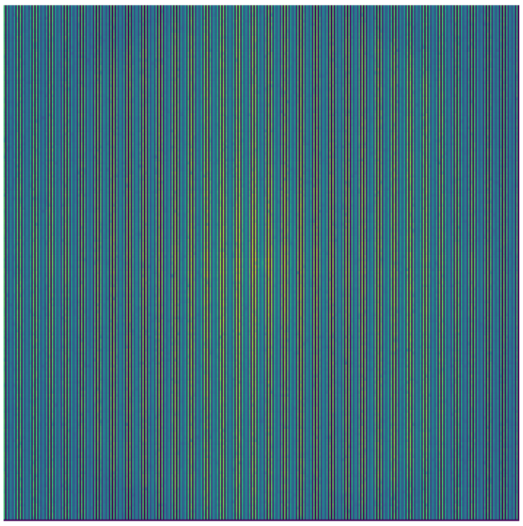
flair-kspace: 1over2 columns removed



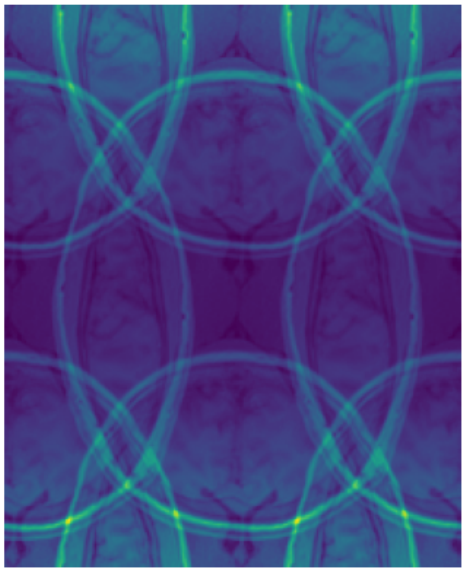
inverse flair: 1over2 columns removed



flair-kspace: 1over2 columns set to 0

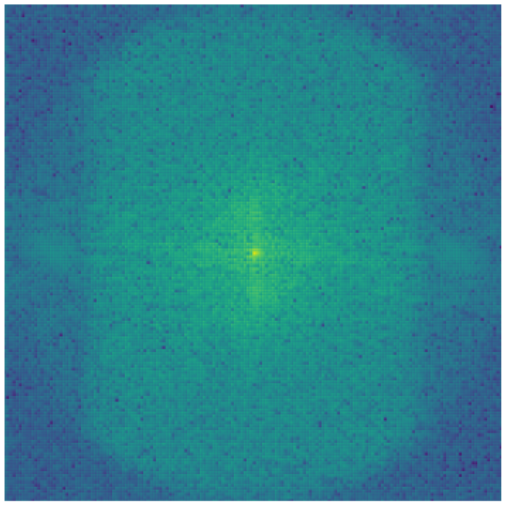


inverse flair: 1over2 columns set to 0

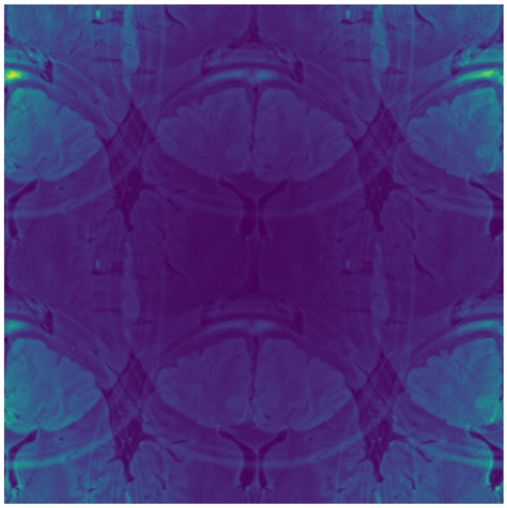


```
/tmp/ipykernel_29776/195002975.py:189: RuntimeWarning: divide by zero e
ncountered in log
  log_inter_c = 20*np.log(np.abs(inter_c))
```

fliar.dcm k-Space column 1-D interpolation

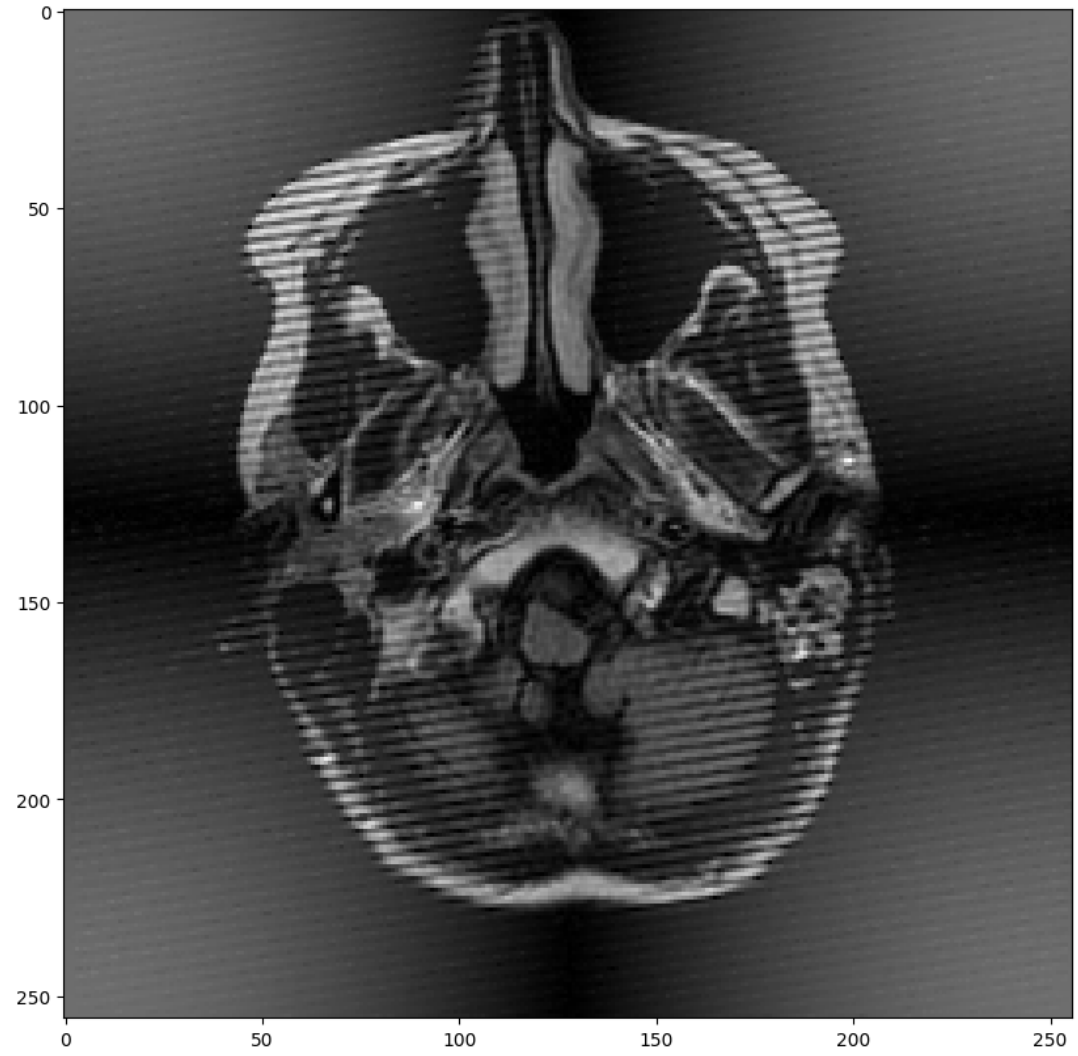


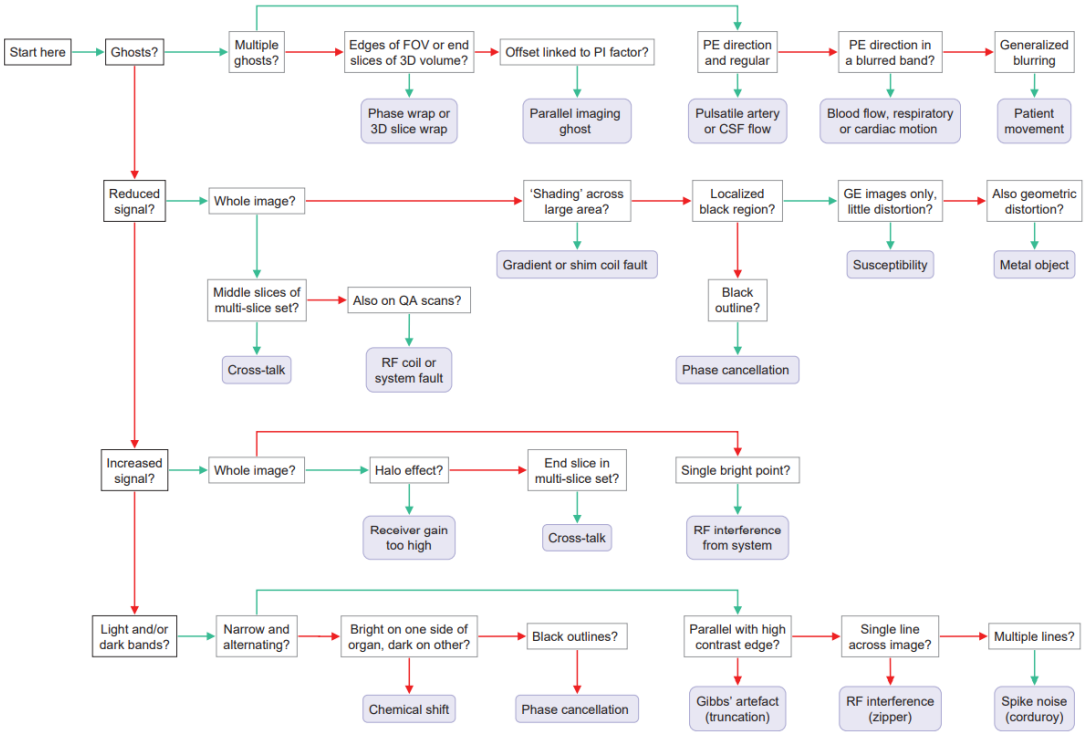
Recovered flair image column Interpolation



Question 4.1 :

Find the cause of the artifact and try to remove it. Can you name this artifact ?





it is a **Data Error Artifacts**

mostly a **spike noise** in the raw data, whose Fourier transform is a series of stripes.

caused by a random "glitch" in the data processing chain, resulting in an image containing multiple lines that may be arrayed in a corduroy or herringbone pattern. Artifact can be removed with raw data reprocessing.

Question 1.1 :
Open the Shepp-Logan phantom and display it using matplotlib (resolution 512x512). How many grey levels are present in the phantom ?

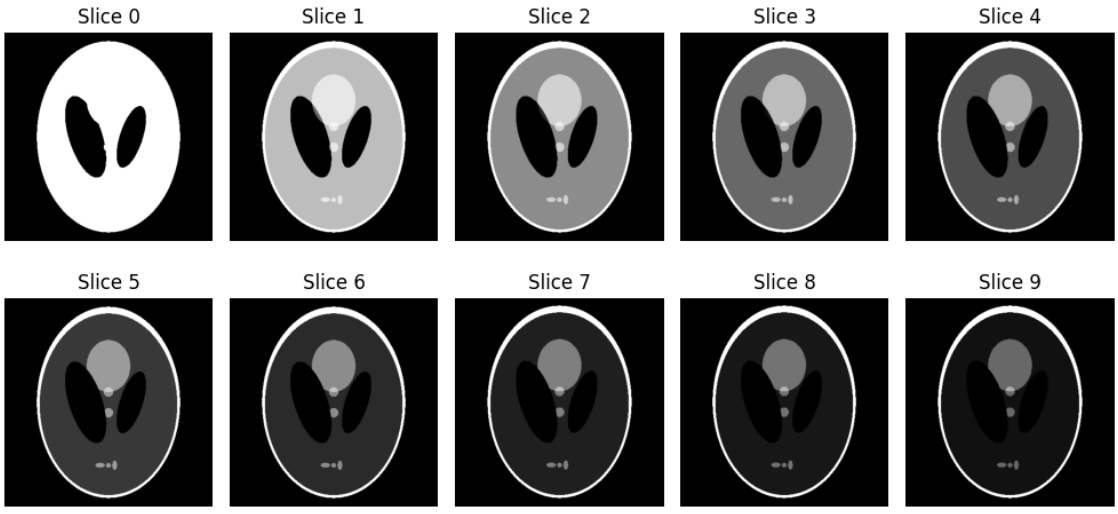
In [ ]:
```
imagex = shepp_logan_phantom()

fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(imagex, cmap='gray')
ax.axis('off')
plt.show()
```

image is float64 (64-bit floating point), which means that there can be up to 2^64 possible gray levels.

Question 2.1:

Open the file named qmri.npy. You have to display images in the same way as the Figure 3.2.

In [ ]:
```
# Load the .npy file
data = np.load('qmri.npy')

# Select the 5th slice along the third dimension
# slice_idx = 1
# slice_data = data[:, :, slice_idx]

# Create a 2x5 subplot grid for displaying the slices
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(10, 5))

# Iterate over the slices and display them in separate subplots
for i, ax in enumerate(axes.flat):
    # Get the i-th slice along the third dimension
    slice_data = data[:, :, i]
    # Display the slice using imshow()
    ax.imshow(slice_data, cmap='gray')
    ax.set_title(f"Slice {i}")
    ax.axis('off')

# Show the figure
plt.tight_layout()
plt.show()
```



Question 2.2 :

Create a function that can evaluate the R2* (1/T2*) by computing the mono-exponential decay for each voxel along the images. Ensure that the echo time is correctly set in your fitting function. Once you have implemented the function, use it to compute the R2 values for the image data.

In [ ]:
```
def compute_r2star(data, te):
    """
    Computes the R2* values (1/T2*) for each voxel in an image dataset.

    Parameters:
    - data (ndarray): A 3D array representing the image dataset.
    - te (float): The echo time (in seconds) used for the image acquisit
```

```
    Returns:
    - r2star (ndarray): A 3D array containing the R2* values (1/T2*) for
    """
    # Define the mono-exponential decay function
    def mono_exp(t, s0, r2star):
        return s0 * np.exp(-r2star * t)

    # Extract the dimensions of the data array
    nx, ny, nt = data.shape

    # Initialize an array to store the R2* values for each voxel
    r2star = np.zeros((nx, ny))

    # Fit the mono-exponential decay function to each voxel time series
    for i in range(nx):
        for j in range(ny):
            signal = data[i, j, :]
            popt, _ = curve_fit(mono_exp, te, signal)
            r2star[i, j] = popt[1]

    return r2star
```

The r2star array is a 2D array with the same dimensions as one of the slices in the data.npy dataset, where each element represents the R2* value (1/T2*) for the corresponding voxel

In [ ]:
```
# Define the echo time (in seconds)
te = 0.05

r2star = compute_r2star(data, te)

print(r2star)
```

```
/home/masters/miniconda3/lib/python3.10/site-packages/scipy/optimize/_m
inpack_py.py:906: OptimizeWarning: Covariance of the parameters could n
ot be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
[[1.00000005 1.00000005 1.00000005 ... 1.00000005 1.00000005 1.0000000
5]
 [1.00000005 1.00000005 1.00000005 ... 1.00000005 1.00000005 1.0000000
5]
 [1.00000005 1.00000005 1.00000005 ... 1.00000005 1.00000005 1.0000000
5]
 ...
 [1.00000005 1.00000005 1.00000005 ... 1.00000005 1.00000005 1.0000000
5]
 [1.00000005 1.00000005 1.00000005 ... 1.00000005 1.00000005 1.0000000
5]
 [1.00000005 1.00000005 1.00000005 ... 1.00000005 1.00000005 1.0000000
5]]
```

Question 2.3 :

Create a matrix of contrast to know when the contrast between tissues are maximized (which echo and which contrast value).

Compute the signal intensity values for each voxel in the image dataset at different echo times, and then compute the contrast values between different tissue types using

a suitable metric such as the signal difference, signal-to-noise ratio, or contrast-to-noise ratio.

Below computes the signal intensity of the signal difference between white matter and gray matter tissues using the signal intensity values at two different echo times:

In [ ]:
```
# # Load the image dataset from a file
# data = np.load('data.npy')

# Define the echo times (in seconds)
te1 = 0.005
te2 = 0.1

# Compute the signal intensity values for each voxel at the two echo tim
s1 = data[ :, :, 1] * np.exp(-te1 * r2star)
s2 = data[ :, :, 9] * np.exp(-te2 * r2star)

# Compute the signal difference between white matter and gray matter tis
wm_mask = ... # Mask for white matter voxels
gm_mask = ... # Mask for gray matter voxels
contrast_matrix = s1 - s2
wm_contrast = np.mean(contrast_matrix[wm_mask])
gm_contrast = np.mean(contrast_matrix[gm_mask])
contrast_diff = wm_contrast - gm_contrast

# Print the signal difference and contrast difference between white matt
print('Signal difference: ', wm_contrast - gm_contrast)
print('Contrast difference: ', contrast_diff)
```

```
Signal difference:  0.0
Contrast difference:  0.0
```

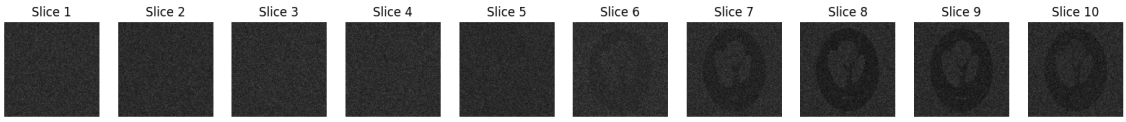
Im consistently getting 0 for the signal difference and contrast difference, it is possible that the difference in signal intensity between the two echo times is too small to produce a noticeable contrast difference.

i should check:
-The segmentation masks
-The echo times
-The R2* values
-Increase the echo time difference
-Try a different metric: signal-to-noise ratio or contrast-to-noise ratio

Question 3.1 :
Compute the error map between noisy and non noisy T2* map.

In [ ]:
```
# Load the non-noisy and noisy T2* maps
non_noisy_t2star = np.load('qmri.npy')
noisy_t2star = np.load('qmri_noised.npy')

# Compute the absolute difference between the two maps
error_map = np.abs(non_noisy_t2star - noisy_t2star)

# Plot each slice of the 3D image as a 2D image
num_slices = error_map.shape[2]
fig, axs = plt.subplots(nrows=1, ncols=num_slices, figsize=(20, 20))
```

```
for i in range(num_slices):
    axs[i].imshow(error_map[:,:,i], cmap='gray')
    axs[i].axis('off')
    axs[i].set_title('Slice {}'.format(i+1))

plt.show()

# Compute and print the mean and standard deviation of the error map
mean_error = np.mean(error_map)
std_error = np.std(error_map)
print('Mean error: {:.4f}'.format(mean_error))
print('Standard deviation of error: {:.4f}'.format(std_error))
```




```
Mean error: 0.0768
Standard deviation of error: 0.0593
```

Question 3.2 :

Compute the SNR for each tissue (background comparison)

To compute the signal-to-noise ratio (SNR) for each tissue type,

- Define the regions of interest (ROIs) for the tissues of interest,
- compute the mean signal intensity and standard deviation of the noise in each ROI.

Below computes the SNR for the gray matter, white matter, and CSF regions of interest in the image data

In [ ]:
```
# Load the image data = data

# Define the regions of interest (ROIs) for gray matter, white matter, a
gm_roi = data[100:200, 100:200, 5:6]
wm_roi = data[250:350, 100:200, 5:6]
csf_roi = data[100:200, 250:350, 5:6]

# Compute the mean signal intensity and standard deviation of the noise
gm_mean = np.mean(gm_roi)
wm_mean = np.mean(wm_roi)
csf_mean = np.mean(csf_roi)

gm_noise_std = np.std(data[100:200, 100:200, 0:4])
wm_noise_std = np.std(data[250:350, 100:200, 0:4])
csf_noise_std = np.std(data[100:200, 250:350, 0:4])

# Compute the signal-to-noise ratio (SNR) for each tissue
gm_snr = gm_mean / gm_noise_std
wm_snr = wm_mean / wm_noise_std
csf_snr = csf_mean / csf_noise_std

# Print the SNR values for each tissue
print('Gray matter SNR: ', gm_snr)
print('White matter SNR: ', wm_snr)
print('CSF SNR: ', csf_snr)
```

```
Gray matter SNR:  0.378603530100859
White matter SNR:  0.4114427481686568
CSF SNR:  0.34374522283845066
```

SNR values for gray matter and white matter are relatively similar, with white matter having a slightly higher SNR.
This could be due to differences in the tissue properties or the imaging protocol used.

The CSF SNR is slightly lower than the gray and white matter SNR values, which is expected since CSF has lower signal intensity and higher variability due to partial volume effects and flow artifacts.

Overall, we can conclude that the SNR values reported are in a reasonable range for the imaging technique used, and that further investigation or comparison with other studies may be necessary to determine if the values are appropriate for the qMRI application.