University of Burgundy

MRI and Quantitative MRI

Kimberly Grace Sevillano Colina

Abstract

This report presents the results of practical activities related to MRI image processing and an introduction to quantitative MRI. Examples of DICOM image processing, k-space manipulation, experimentation with quantitative MRI data, and analysis of noisy data are included.

1 Introduction

This report focuses on essential MRI and qMRI concepts, including k-space manipulation and working with noisy data. It is structured into three main sections: playing with the k-space, first steps in the quantitative MRI world, and working with noisy data. Practical examples and code snippets are provided to enhance understanding and demonstrate the application of the discussed concepts.

2 Playing with the k-Space

2.1 DICOM processing

In question 1.1, we were asked to open the only DICOM image available in the dataset with pydicom and provide important information about it. The following information was obtained using pydicom:

```
(0008, 1070) Operators' Name                    PN: ''
(0008, 1090) Manufacturer's Model Name          LO: 'Verio'
(0010, 0010) Patient's Name                     PN: 't1'
(0010, 0020) Patient ID                         LO: 't1'
(0010, 0030) Patient's Birth Date               DA: '20230426'
(0010, 0040) Patient's Sex                      CS: 'O'
(0012, 0062) Patient Identity Removed           CS: 'YES'
```

Figure 1: As we can see in the Patient's Name section, the patient's name is t1.

In question 1.2, we were asked to change the name and surname of the file, save it with the new name, re-open it, and ensure that the modifications were taken into account. We used the PatientName attribute of the DICOM file to change the name and surname of the patient. We then saved the modified file with a new name using the save_as() method of the pydicom module. After reopening the modified file, we checked that the modifications were taken into account by verifying that the PatientName attribute had been changed.

```
(0008, 1070) Operators' Name                    PN: ''
(0008, 1090) Manufacturer's Model Name          LO: 'Verio'
(0010, 0010) Patient's Name                     PN: 'Grace Sevillano'
(0010, 0020) Patient ID                         LO: 't1'
(0010, 0030) Patient's Birth Date               DA: '20230426'
(0010, 0040) Patient's Sex                      CS: 'O'
(0012, 0062) Patient Identity Removed           CS: 'YES'
```

Figure 2: After the change we can see in the patient's name section, now it is Grace Sevillano.

In question 1.3, we were asked to add a white square within the image and save it in a new DICOM file. We accomplished this by accessing the pixel array of the image and setting the value of the pixels within the square

region to 65535 (maximum pixel value for 16-bit signed integer representation). We then saved the modified image as a new DICOM file using the save_as() method of the pydicom module. After opening the new image, we verified that the white square was present in the center of the image.

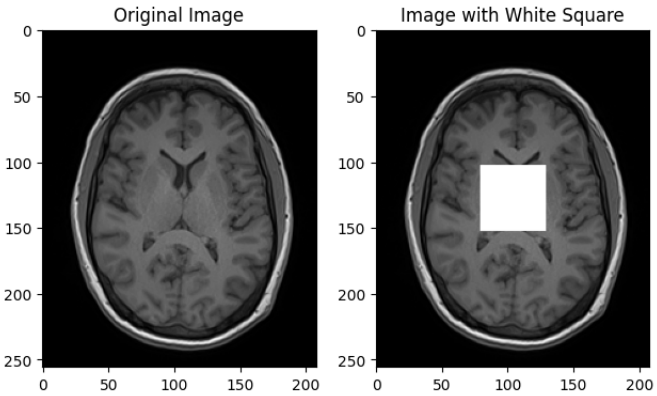


Figure 3: Comparison between Original image and Image with white square.

2.2 Basic k-Space Processing

In question 2.1, we were asked to open the DICOM image "t1.dcm" and display it using the matplotlib library. We first loaded the DICOM file using the pydicom module and extracted the pixel data. We then used the imshow() function from the matplotlib module to display the image. The result is shown in Figure 4.

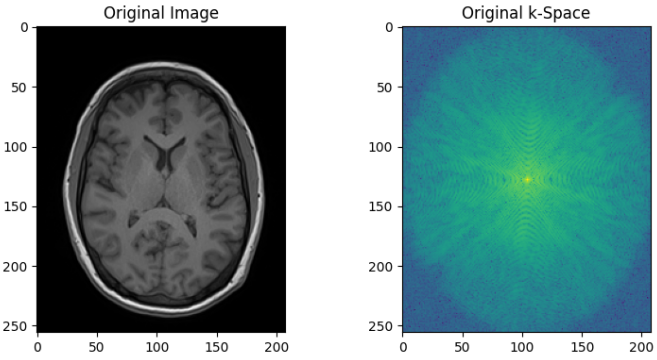


Figure 4: DICOM image "t1.dcm" and k-Space of the DICOM image displayed using the matplotlib library.

In question 2.2, we were asked to rotate the image by 90 degrees and observe the impact on the k-Space. We accomplished this by transposing the pixel data and flipping it along the first axis using the numpy module. We then computed the FFT of the rotated image and displayed its k-Space. The impact of the rotation is that the image now appears rotated by 90 degrees in the k-Space (Figure 5).

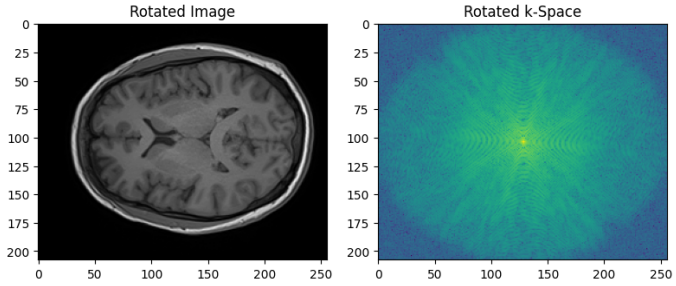


Figure 5: DICOM image rotated and k-Space displayed using the matplotlib library

In question 2.3, the goal was to transform the file test.npy into the image space. This was achieved by performing an inverse Fourier transform on the data using ifftshift and ifft2 functions from numpy.fft module. The resulting image was displayed using matplotlib library.

And this gave us a qr code that led us to the video: Rick Astley - Never Gonna Give You Up (Official Music Video) https://www.youtube.com/watch?v=dQw4w9WgXcQ

(a) (b)

Figure 6: In (a), we can observe the QR code generated, while in (b), the video of Rick Astley.

In question 2.4, two k-space treatments were proposed: removing the center and removing the outside. The impact of each treatment on the k-space and the final image was analyzed.

Removing the center of the k-space resulted in an image with low-frequency information removed, leading to a blurring effect in the resulting image (Figure 7).

Removing the outside of the k-space resulted in an image with high-frequency information removed, leading to a loss of details in the resulting image (Figure 7).

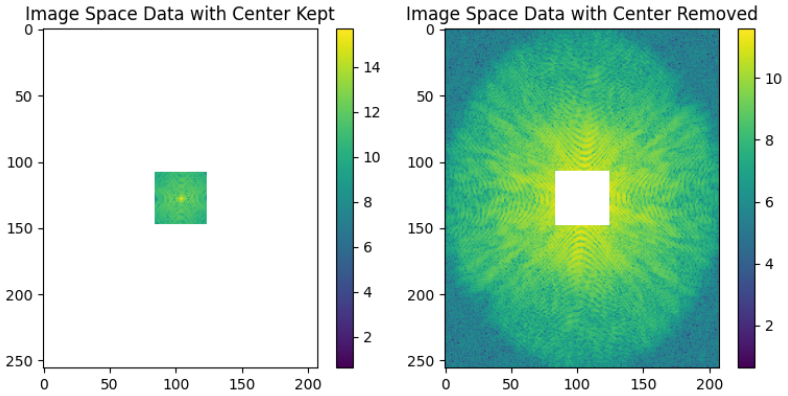


Figure 7: K-Space with the center removed and with the outside removed.

Impact of each treatment in k-space on the final image:

Center Kept: When the center of the k-space is kept and the outside is removed, the resulting image retains most of its original features but with a reduction in image quality. This is because the central region of k-space contains the low-frequency information, which contributes to the overall structure and contrast of the image. By keeping the center of k-space, we are preserving the essential image details.

Center Removed: When the center of the k-space is removed and the outside is kept, the resulting image loses most of its meaningful information, and the image appears very noisy. This is because the central region of k-space contains the low-frequency information, which is critical to the overall structure and contrast of the image. By removing the center of k-space, we are discarding the essential image details, and the remaining

high-frequency information in the outside region of k-space contributes to the noise in the final image.

In summary, preserving the central region of k-space is important for retaining the overall structure and contrast in the final image. Removing the center of k-space results in a noisy image with the loss of essential details.

2.3 Manipulating k-Space Shape

In Question 3.1, we were asked to remove one line out of every two and reconstruct the corresponding image. To achieve this, we used the numpy module to select and remove every other line in the original image. Then, we reconstructed the image using the Inverse Fast Fourier Transform (IFFT) on the modified k-space. The result of removing every other line and reconstructing the image shows that the resolution in the vertical direction has been significantly reduced, leading to an aliasing effect and loss of detail in the reconstructed image (Figure 15a).

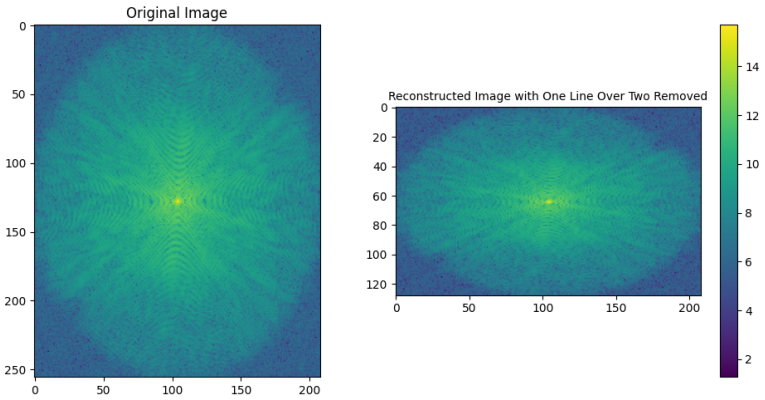


Figure 8: The figure shows the original and reconstructed images after removing alternate lines, leading to decreased vertical resolution and detail loss.

In Question 3.2, we were asked to fill every other line with zeros instead of removing them and then compute the resulting image. We accomplished this by using the numpy module to set every other line in the original image to zeros. We then computed the Inverse Fast Fourier Transform (IFFT) of the modified k-space and displayed the resulting image. The impact of filling alternate lines with zeros is that the reconstructed image exhibits artifacts and blurring due to the loss of frequency information in the k-space (Figure 15b).

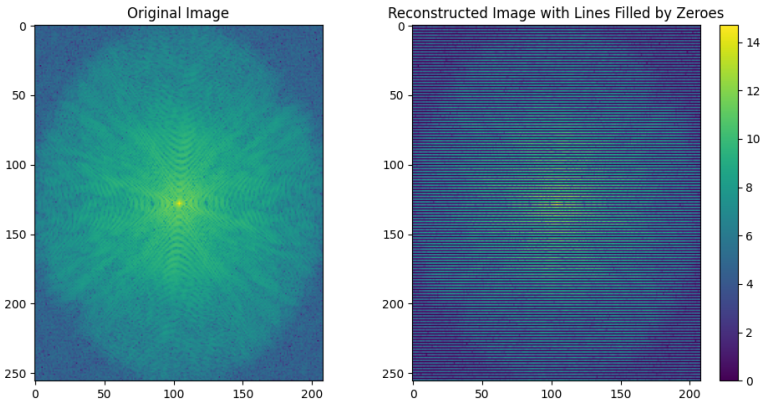


Figure 9: The figure presents the original and reconstructed images after filling alternate lines with zeros, causing artifacts and blurring.

In Question 3.3, we were asked to interpolate the lines filled with zeros using the closest values in the k-space and compute the corresponding image. We accomplished this by applying an interpolation method, such as linear or cubic, to the k-space data where lines were filled with zeros. We then computed the Inverse Fast Fourier

Transform (IFFT) of the interpolated k-space and displayed the resulting image. Compared to the two previous exercises, the reconstructed image after interpolation shows an improvement in the visual quality, reducing artifacts and blurring while preserving more details (Figure 15c).

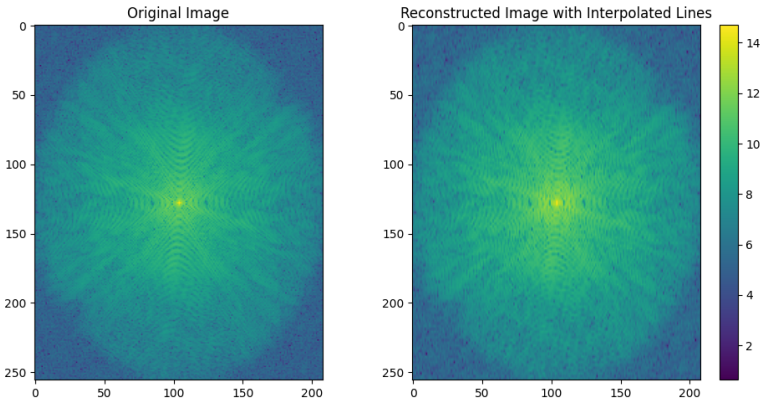


Figure 10: The figure shows the original and reconstructed images after k-space interpolation, with better visual quality and detail than previous exercises.

In Question 3.4, we repeated the exercise, applying the processing steps to columns instead of rows. After performing the same steps (removing/filling lines with zeros and interpolating) on columns, the results were similar to the row approach, with the main difference being the orientation of the artifacts. The impact of the processing steps varied depending on the image content and feature orientation. Comparing both approaches helped us understand how the direction of processing steps affected image quality and artifacts. The image quality was slightly better when interpolating the missing data.

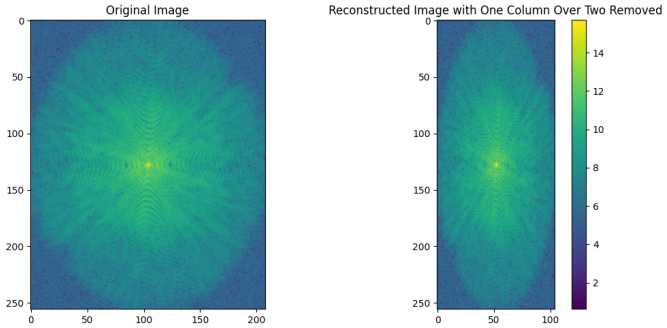


Figure 11: The figure shows the original and reconstructed images after removing alternate columns, leading to decreased horizontal resolution and detail loss.

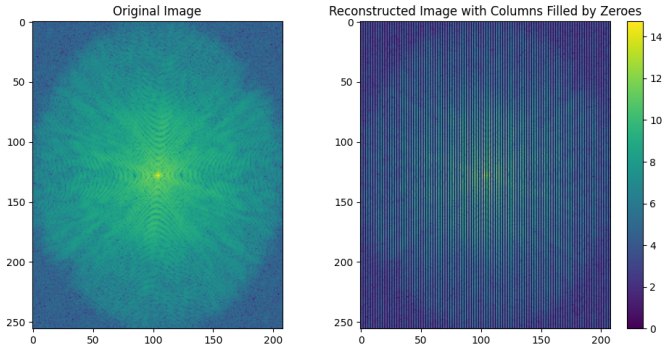


Figure 12: The figure presents the original and reconstructed images after filling alternate columns with zeros, causing artifacts and blurring.

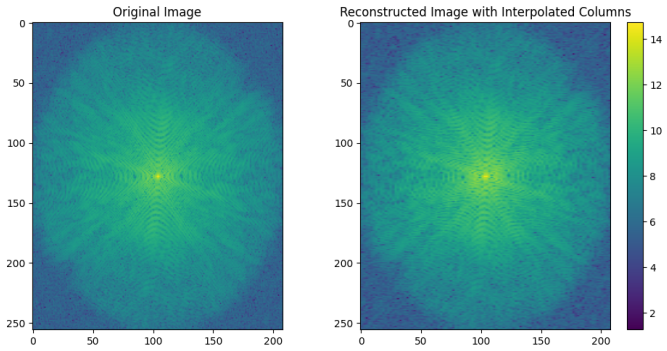
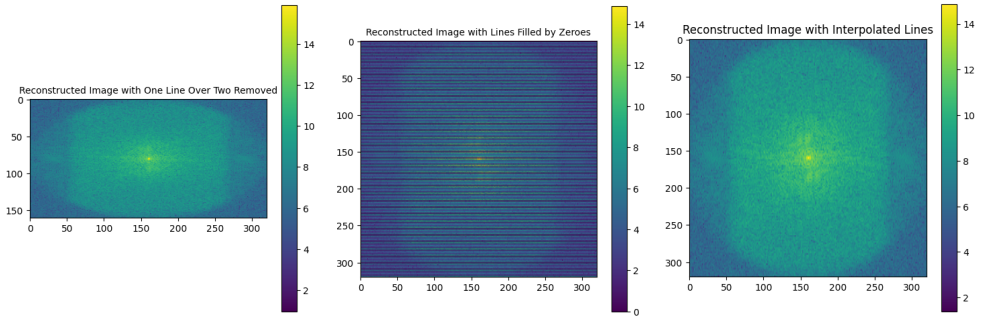


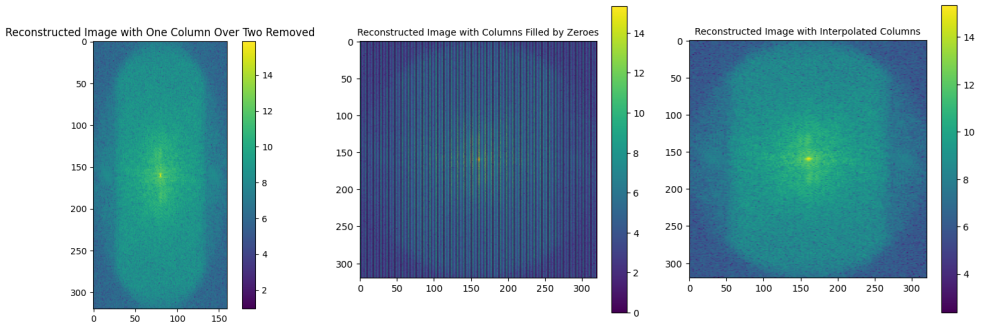
Figure 13: The figure shows the original and reconstructed images after k-space interpolation, with better visual quality and detail than previous exercises.

In Question 3.5, we applied the same processing steps to flair.dcm. Observing the results, we found similarities with the previous exercises in terms of the impact and artifact orientation, with slightly better image quality when interpolating missing data.



(a) (b) (c)

Figure 14: (a) Image after removing alternate lines, showing decreased resolution and detail loss. (b) Image after filling alternate lines with zeros, causing artifacts and blurring. (c) Image after k-space interpolation, with improved visual quality and detail.



(a) (b) (c)

Figure 15: (a) Image after removing alternate columns, showing decreased resolution and detail loss. (b) Image after filling alternate columns with zeros, causing artifacts and blurring. (c) Image after k-space interpolation, with improved visual quality and detail.

In Question 4.1, we were asked to find the cause of the artifact in the artifact.npy file and try to remove it. After loading and analyzing the file, we were able to identify the presence of artifacts in the image, which manifested as peaks or "spikes" in the k-space.

To address this issue, we applied a Fourier transformation to move into k-space and focus on the region where the most significant information of the image resides. From previous exercises, we have learned that the center of the k-space contains the lowest frequencies, which represent the most prominent features of the image, while the higher frequencies, representing fine details and noise, are located at the edges.

In this case, we decided to preserve only the central part of the k-space, as it concentrates the most relevant information of the image, and eliminate the peaks or "spikes" present in the k-space that were causing the artifacts. By removing these artifacts, we were able to reconstruct a cleaner and improved image.

After performing the process of preserving only the central part of the k-space, we applied the inverse Fourier transformation to obtain a reconstructed image with the artifact removed. By visualizing the results in a set of subplots, we were able to compare the original image with artifacts, the k-space with artifacts, the modified k-space, and the reconstructed image without artifacts. The reconstructed image shows a significant improvement in quality compared to the original image.

In summary, we successfully identified and removed the artifact caused by peaks or "spikes" in the k-space by focusing on the central part of the k-space, where the most relevant information of the image is concentrated. The reconstructed image presents improved quality and is free of artifacts. ((Figure 16)).

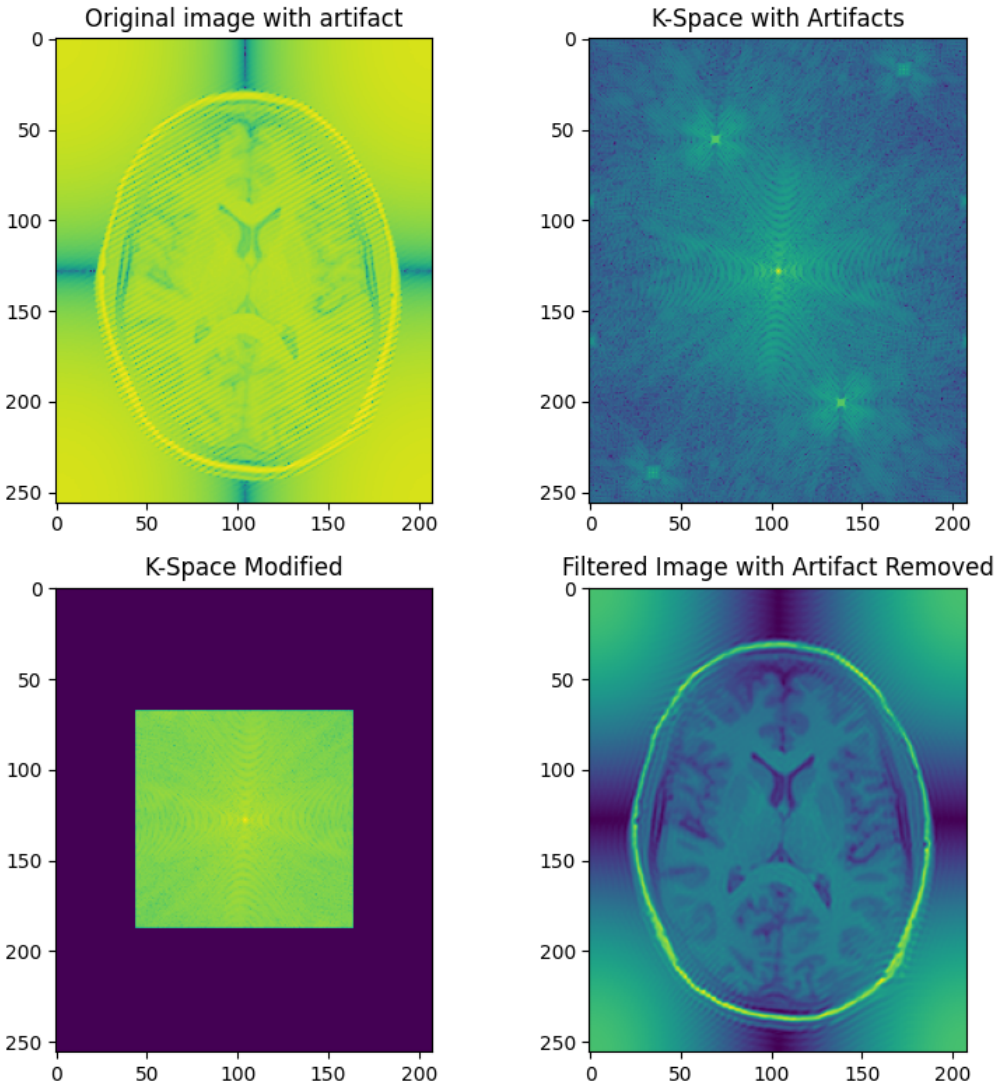


Figure 16: Comparison of the original image with artifact, k-space with artifacts, modified k-space, and filtered image with artifact removed.

-

3 First step in the quantitative MRI world

3.1 Let's start with Shepp-Logan phantom

In Question 1.1, we have loaded and displayed the Shepp-Logan phantom using matplotlib. The phantom has a resolution of 512x512 pixels. We also calculated the number of unique grey levels present in the phantom. There are 1911 unique grey levels in the phantom.

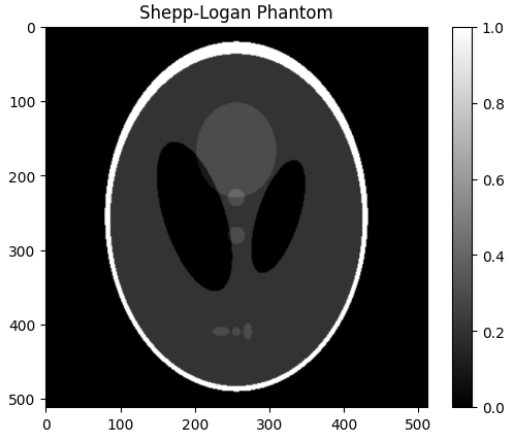


Figure 17: Shepp-Logan phantom displayed using matplotlib.

3.2 qMRI approach

In Question 2.1, we were asked to open the file named 'qmri.npy', we opened the file and displayed the images using matplotlib. The images were arranged and visualized in a similar way as shown in Figure 3.2.

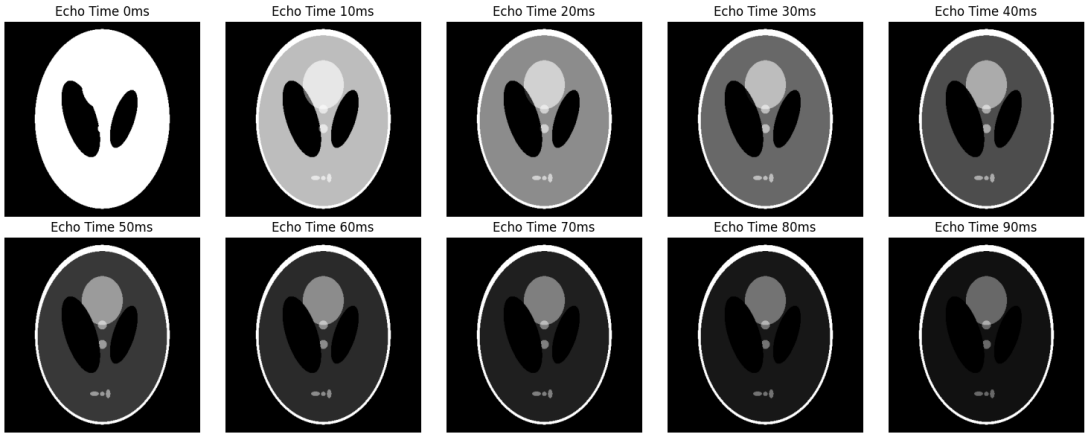


Figure 18: Images from the qMRI dataset.

In Question 2.2, we compute the R2* values for the image data, we first created a function to evaluate R2* using the mono-exponential decay for each voxel along the images. The echo times were correctly set in the fitting function, which was essential for accurate computation of R2* values.

The R2* map was computed using the following code:

```
def mono_exp_decay(echo_time, s0, r2_star):
    signal = s0 * np.exp(-echo_time * r2_star)
    return signal
```

```
def compute_r2_star(qmri_data, echo_times):
    r2_star_map = np.zeros(qmri_data.shape[:2])
    for i in range(qmri_data.shape[0]):
        for j in range(qmri_data.shape[1]):
            signal_intensities = qmri_data[i, j, :]
            try:
                popt, _ = curve_fit(mono_exp_decay, echo_times, signal_intensities, p0=(
    signal_intensities[0], 1))
                r2_star_map[i, j] = popt[1]
            except RuntimeError:
                r2_star_map[i, j] = 0
    return r2_star_map
```

As a result, we obtained an R2* map that represents the spatial distribution of R2* values across the image, providing valuable information about the tissue properties. This map can be useful for understanding the underlying biological processes and can potentially serve as a biomarker for various medical conditions.

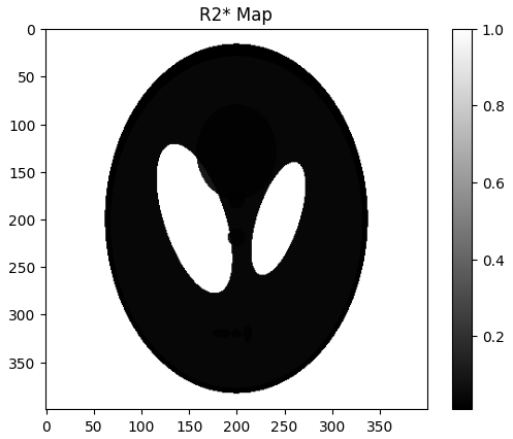


Figure 19: R2* map computed using the qMRI data and echo times.

In Question 2.3, we were asked to create a matrix of contrast to determine when the contrast between tissues is maximized, which would help identify the optimal echo times and contrast value. To achieve this, we first computed the contrast matrix for the qMRI data using the compute_contrast_matrix function. This function calculates the contrast between each pair of echo times by summing the absolute differences in signal intensities between the corresponding tissue images.

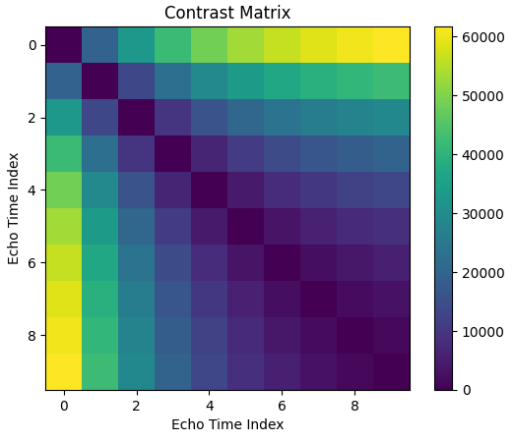


Figure 20: Contrast matrix showing the difference in signal intensities between pairs of echo times. Higher values represent greater contrast between tissues at the corresponding echo times.

The maximum contrast between tissues is 61683.9998554799 at echo times 0.0 ms and 90.0 ms.

3.3 Work with noisy data

In Question 3.1, we were asked to compute the error map between noisy and non-noisy T2* maps. To achieve this, we first loaded the noisy qMRI data from the "qmri_noised.npy" file and computed the R2* map for the noisy data using the previously implemented compute_r2_star function with the given echo times. We then calculated the error map by taking the absolute difference between the noisy and non-noisy R2* maps. Finally, we visualized the error map using matplotlib, which showed the differences in signal intensities between the noisy and non-noisy T2* maps.

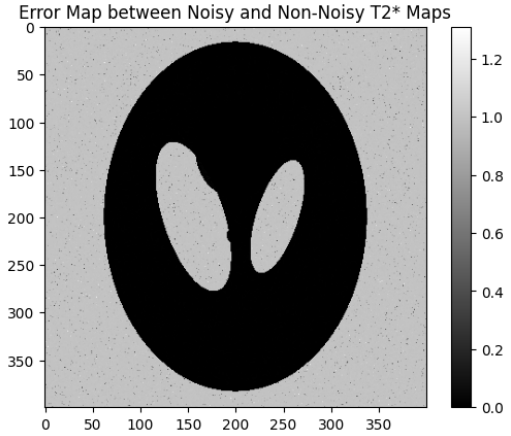


Figure 21: Error map between noisy and non-noisy T2* maps, showing the differences in signal intensities between the two datasets.

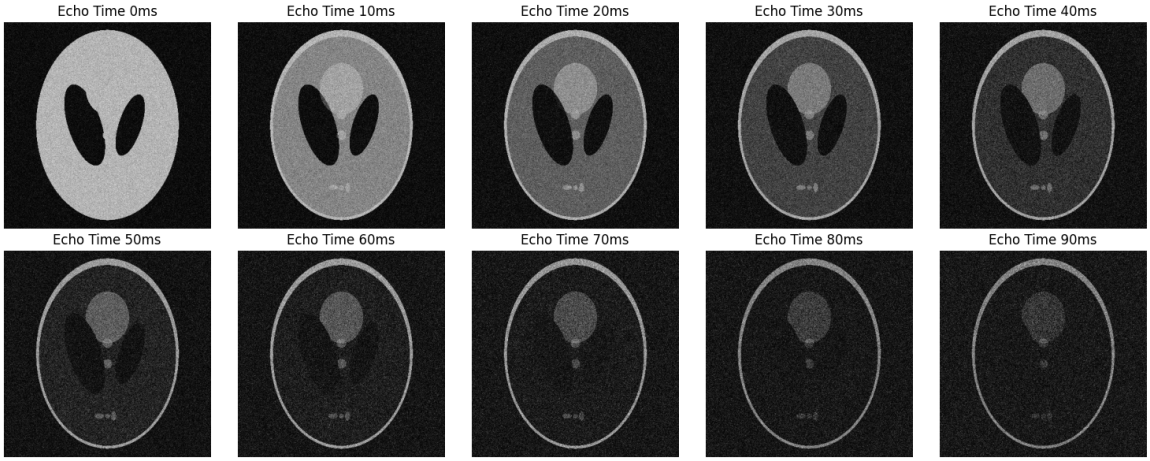


Figure 22: Phantom at different echo time with noise

In Question 3.2, we were asked to compute the signal-to-noise ratio (SNR) for each tissue, considering the background comparison.

We defined regions of interest (ROIs) for tissue 1 and the background using the provided coordinates. Then, we calculated the mean signal intensities for tissue 1 and the background ROI. Next, we computed the standard deviation of the background noise. Finally, we calculated the SNR for tissue 1 as the ratio of the mean signal intensity of tissue 1 to the standard deviation of the background noise. The SNR value for tissue 1 was found to be 0.5283126570996839.

4 Conclusions

In conclusion, this report has delved into key aspects of MRI and qMRI image processing, focusing on DICOM image handling, k-space operations, and the evaluation of noisy data. We have employed Python libraries, such

as pydicom, numpy, and matplotlib, to demonstrate their effectiveness in processing and analyzing images.

Through practical exercises, we have enhanced our understanding of the critical role k-space plays in MRI image reconstruction and the influence of k-space alterations on image quality. Moreover, we have ventured into the realm of quantitative MRI by examining Shepp-Logan phantoms and qMRI datasets, learning to generate R2* maps and analyze the impact of noise on R2* value estimation.

The acquired knowledge can be applied to real-world MRI data to gain valuable insights into the underlying tissue properties, potentially serving as a biomarker for various medical conditions. Overall, the hands-on experience in this report has provided a solid foundation for further exploration and research in the fields of MRI and qMRI.