

Exercise 1

a)

```
In [2]: from PIL import Image
from IPython.display import display
img = Image.open('EX01_a.jpeg')
display(img,)
```

$$E = -\mu B_0 \quad (1) \quad \mu = I\gamma \quad (2) \quad r = \frac{n\downarrow}{n\uparrow} \quad (3) \quad I = \pm \frac{1}{2}\hbar \quad (4)$$

$$\text{from (1) \& (2)} \Rightarrow E = -I\gamma B_0 \quad (5)$$

$$\text{from (4) \& (5)} \rightarrow I_{\uparrow} = +\frac{1}{2}\hbar \rightarrow E_{\uparrow} = -\frac{1}{2}\hbar\gamma B_0 \quad (6)$$

$$\rightarrow I_{\downarrow} = -\frac{1}{2}\hbar \rightarrow E_{\downarrow} = +\frac{1}{2}\hbar\gamma B_0 \quad (6)$$

given $\gamma > 0$ for hydrogen,
the low-energy (parallel) state
becomes the $E = -\frac{1}{2}\hbar\gamma B_0$
and this is how the signs $\uparrow\downarrow$
were assigned

$$\text{from } n = n\uparrow + n\downarrow \text{ and (3):}$$

$$n\downarrow = r n\uparrow \rightarrow n = n\uparrow + r n\uparrow \rightarrow n = n\uparrow(1+r) \rightarrow n\downarrow = \frac{n}{1+r}$$

$$\rightarrow n\downarrow = \frac{rn}{1+r}$$

Now we count the number of uncancelled number of spins:

$$n\uparrow - n\downarrow = \frac{n}{1+r} - \frac{rn}{1+r} = \frac{(1-r)n}{1+r}$$

$$\text{each uncancelled spin contributes } \mu = \frac{1}{2}\hbar\gamma \quad \text{so} \quad M_0 = \frac{(1-r)n}{1+r} \cdot \frac{\hbar\gamma}{2}$$

Substituting values:

$$r = e^{-\frac{\Delta E}{k_B T}} \rightarrow T = 310 \text{ K}$$

$$\rightarrow K_B = 1,381 \text{ e}^{-23} \text{ J/K}$$

$$\rightarrow \Delta E = \frac{1}{2}\hbar\gamma B_0 + \frac{1}{2}\hbar\gamma B_0 = \hbar\gamma B_0 \rightarrow \hbar = 1,054 \text{ e}^{-34} \text{ Js}$$

$$\rightarrow B_0 = 1,5 \text{ T} \rightarrow \gamma = 267,613 \text{ e}^6 \text{ rad/T}$$

$$\Delta E = 1,054 \text{ e}^{-34} \cdot 1,5 \cdot 267,613 \text{ e}^6 \approx 9,23 \text{ e}^{-26} \text{ J}$$

$$\frac{-\Delta E}{K_B T} = \frac{9,23 \text{ e}^{-26}}{1,381 \text{ e}^{-23} \cdot 310} \approx 9,89 \text{ e}^{-6}$$

$$r = e^{-9,89 \text{ e}^{-6}} \approx 0,999 \rightarrow \text{I avoid approximation to 1 to avoid multiplication by 0 in } M_0.$$

$$\mu = \frac{1}{2}\hbar\gamma = \frac{1}{2} \cdot 1,054 \text{ e}^{-34} \cdot 267,613 \text{ e}^6 \approx 1,41 \text{ e}^{-26} \text{ J/T}$$

$$M_0 = \frac{(1-0,999)}{(1+0,999)} 6,69 \text{ e}^{-19} \cdot 1,41 \text{ e}^{-26} \approx 4,66 \text{ e}^{-12} \text{ J/T}$$

b)

```
In [5]: from PIL import Image
from IPython.display import display
img = Image.open('EX01_b.jpeg')
display(img,)
```

$$b) r = e^{\frac{e^f B_0}{k_B T}} \quad \xi = -\frac{k_B T}{2} \tanh\left(\frac{e^f B_0}{2}\right)$$

$$M_0 = \left[\frac{1 - e^{\frac{e^f B_0}{k_B T}}}{1 + e^{\frac{e^f B_0}{k_B T}}} \right] \cdot \frac{n k_B T}{2} = -\tanh\left(\frac{e^f B_0}{2}\right) \cdot \frac{n k_B T}{2}$$

$$\frac{dM_0}{dB_0} = -\frac{n k_B T}{2} \operatorname{sech}^2\left(\frac{e^f B_0}{2}\right) \frac{e^f}{2}$$

Taylor series :

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \dots$$

we don't need this

around $B_0 = 0$:

$$M(B_0) = M(0) + (B_0 - 0) \frac{dM_0}{dB_0} \Big|_{B_0=0}$$

$$M(0) = -\tanh(0) \cdot \frac{n k_B T}{2} = 0$$

$$\frac{dM_0}{dB_0} \Big|_{B_0=0} = -\frac{n k_B T}{2} \cdot \operatorname{sech}^2(0) = -\frac{n k_B T}{2} = \frac{n (k_B T)^2}{4 k_B T}$$

$$M(B_0) \approx \frac{n (k_B T)^2}{4 k_B T} B_0 \rightarrow \text{we can see that it is linear in } B_0$$

Some common factors :

$$\frac{k_B T}{2} = M = 7,41 e^{-26} J/T \text{ from part a)} \rightarrow \frac{n k_B T}{2} = 6,69 e^{19} \cdot 7,41 e^{-26} = 9,43 e^{-7} J/T$$

$$\xi = \frac{-k_B T}{K_B T} = \frac{-1,054 e^{-34} \cdot 267,513 e^6}{7,381 e^{-23} \cdot 816} = \frac{-2,82 e^{-26}}{4,28 e^{-21}} = -6,59 e^{-6} J/T$$

note, if $|x| \ll 1 \rightarrow \tanh(x) \approx x$

at $B_0 = 1 T$

$$\text{Exact } M_0 = -\tanh\left(\frac{-6,59 e^{-6}}{2}\right) \cdot 9,43 e^{-7} = +\frac{6,59 e^{-6}}{2} \cdot 9,43 e^{-7} = 3,108 e^{-12} J/T$$

$$\text{approximate } M_0 = \frac{n k_B T}{2} \cdot \frac{k_B T}{K_B T} \cdot \frac{1}{2} \cdot B_0 = 9,43 e^{-7} \cdot 6,59 e^{-6} \cdot \frac{1}{2} = 3,108 e^{-12} J/T$$

basically
- ξ

at $B_0 = 3 T$

$$\text{Exact } M_0 = -\tanh\left(\frac{-6,59 e^{-6} \cdot 3}{2}\right) \cdot 9,43 e^{-7} = \frac{1,977 e^{-5}}{2} \cdot 9,43 e^{-7} = 9,322 e^{-12} J/T$$

$$\text{Approximate } M_0 = 9,43 e^{-7} \cdot 6,59 e^{-6} \cdot \frac{1}{2} \cdot 3 = 9,322 e^{-12} J/T$$

Here looking up to 3 floating point accuracy, the numerical results are identical.

Exercise 2

```
In [6]: from PIL import Image
from IPython.display import display
img = Image.open('IAAN_A2_Ex2a.jpeg')
display(img,)
```

Exercise 2 (Spatial Encoding in MRI)

a) The center frequency at the isocenter will be the Larmor frequency, which can be computed as:

slide 20
PDF 3

$$\omega_0 = \gamma B_0 \rightarrow f_0 = \frac{\gamma B_0}{2\pi} \text{ with } \gamma = 267.513 \times 10^6 \frac{\text{rad}}{\text{sT}}$$

$$\Rightarrow f_c = \frac{267.513 \times 10^6 \cdot 3}{2\pi} \left[\frac{\text{rad}}{\text{sT}} \cdot \frac{\text{T} \cdot 1}{\text{rad}} \right]$$
$$= 127.728 \times 10^6 \text{ Hz}$$
$$f_c = 127.728 \text{ MHz},$$

To compute the bandwidth to excite a 2mm slice we use the following formula

slide 21
PDF 3

$$\Delta z = \frac{2\pi \Delta f}{\gamma G_z} \rightarrow \Delta f = \frac{\Delta z \gamma \cdot G_z}{2\pi}$$
$$\Rightarrow \Delta f = \frac{2 \times 10^{-3} \cdot 267.513 \times 10^6 \cdot 40 \times 10^{-3}}{2\pi} \left[\frac{\text{m} \cdot \text{rad}}{\text{sT}} \cdot \frac{\text{T} \cdot 1}{\text{m} \cdot \text{rad}} \right]$$
$$= 3406.081 \text{ Hz}$$
$$= 3.4061 \text{ kHz},$$

To excite a slice of the same size away from the isocenter we must maintain the same bandwidth Δf , but compute the Larmor frequency at that point, as it changes according to the position, for that we can use:

$$w(z) = \gamma (B_0 + G_z \cdot z) \rightarrow \begin{array}{l} \text{Slide 55 (changed from } x \text{ to } z \\ \text{PDF 3 as the behaviour is equivalent) } \end{array}$$
$$\rightarrow f(z) = \frac{w(z)}{2\pi}$$

```
In [7]: img = Image.open('IAAN_A2_Ex2b.jpeg')
display(img)
```

b) The artifact avoided this way is aliasing, removing artifacts in the frequency encoding direction. By oversampling we increase the FOV within the frequency encoding direction, which allows for a larger image range, ensuring nothing will be "detected" as a low-frequency item after using the low-pass filter, which would cause the aliasing effect (things further ahead, like a nose, appearing behind the nose appears in the back of the head).

Exercise 3

a)

```
In [13]: import numpy as np
import matplotlib.pyplot as plt

# --- 1. Load and Reformat k-space Data ---

# Define the file name
filename = 'mrimage1d.txt'

# Read the text file, flatten it, and 'view' pairs of floats as complex numbers
kspace_data = np.loadtxt(filename).ravel().view(complex).reshape((256, 256))

# --- 2. Reconstruct the Image ---

# The canonical MR reconstruction pipeline:
# 1. np.fft.ifftshift: Undo the "centered" k-space. Moves the center
#   (low frequencies) from (128,128) to the corner (0,0) for the FFT algorithm
# 2. np.fft.ifft2: Perform the 2D Inverse Fourier Transform.
# 3. np.fft.fftshift: Center the resulting image. Moves the image's
#   center (which is at the corner (0,0) after ifft2) back to (128,128).

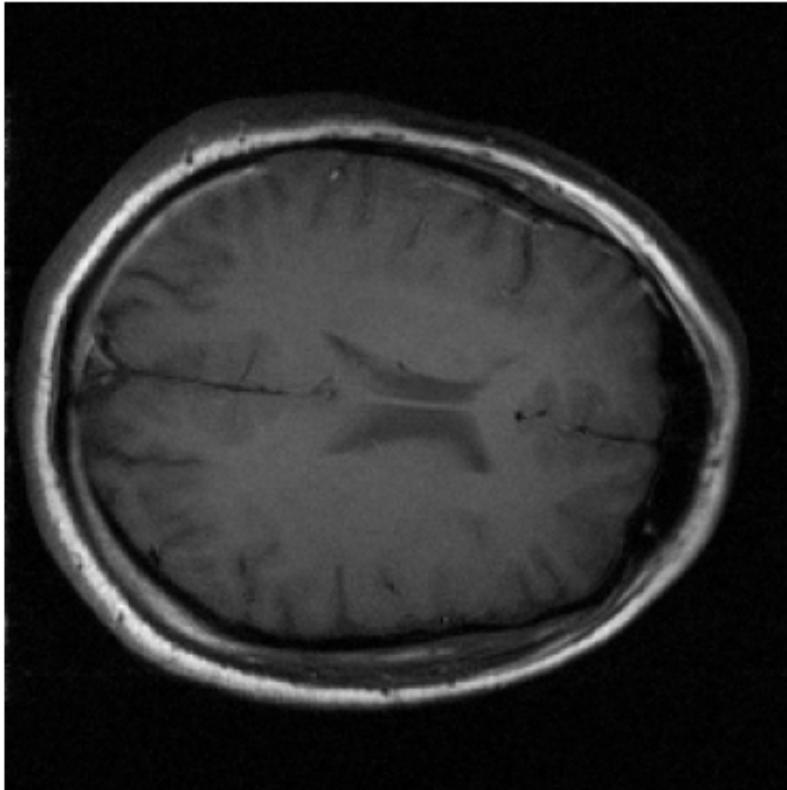
img_complex = np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(kspace_data)))

# --- 3. Visualize the Magnitude Image ---

# Get the magnitude (absolute value) of the complex image
img_magnitude = np.abs(img_complex)

# Display the image
plt.imshow(img_magnitude, cmap='gray')
plt.title('Reconstructed Magnitude Image')
plt.axis('off') # Hide the x and y axes
plt.show()
```

Reconstructed Magnitude Image



b)

```
In [14]: import numpy as np
import matplotlib.pyplot as plt

#2. Prepare Data for Visualization

# k-space: Log-magnitude.
# We use np.log1p(x) which calculates log(1+x)
# This handles x=0 gracefully (log(1)=0) and scales the view.
kspace_log_mag = np.log1p(np.abs(kspace_data))

#Image: Separate real and imaginary parts
img_real = np.real(img_complex)
img_imag = np.imag(img_complex)

#3. Visualize All Three

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))

# Plot 1: Log-Magnitude k-space
ax1.imshow(kspace_log_mag, cmap='gray')
ax1.set_title('k-space (Log-Magnitude)')
ax1.axis('off')

# Plot 2: Real part of the image
ax2.imshow(img_real, cmap='gray')
ax2.set_title('Reconstructed Image (Real Part)')
ax2.axis('off')

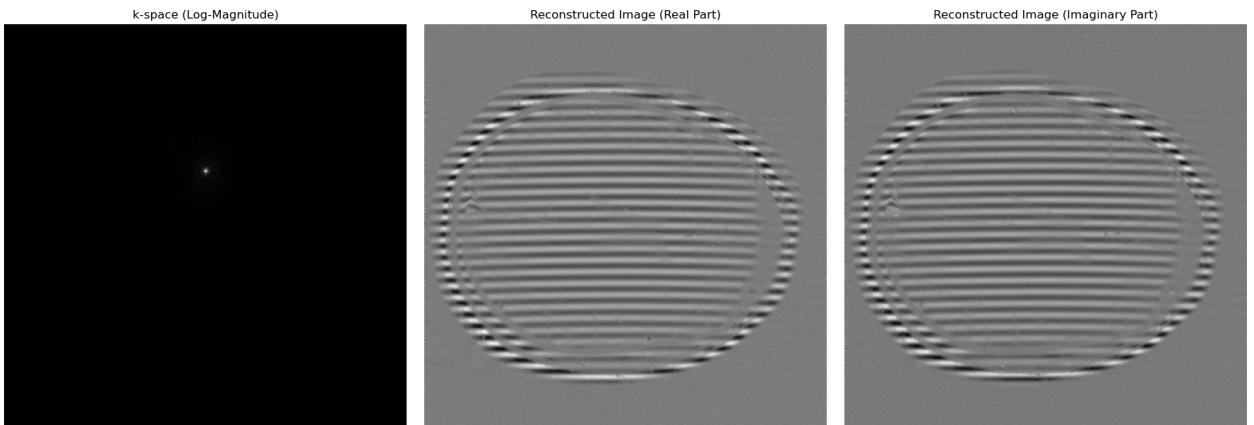
# Plot 3: Imaginary part of the image
```

```

ax3.imshow(img_imag, cmap='gray')
ax3.set_title('Reconstructed Image (Imaginary Part)')
ax3.axis('off')

plt.tight_layout()
plt.savefig('my_reconstruction.png')
plt.show()

```



k-space Visualization (Frequency Domain) Yes, the k-space spectrum is centered as expected. The first plot shows that the highest-magnitude (brightest) values, which represent the low-frequency components of the image, are located in the center of the 256x256 matrix. The magnitudes decay as you move outward toward the high-frequency edges.

Image Visualization (Spatial Domain) Yes, the reconstructed image is real-valued.

Spatial-Frequency Connection This observation is a direct consequence of a fundamental property of the Fourier Transform: A real-valued signal in the spatial domain (the image) must correspond to a frequency-domain signal (k-space) that has conjugate symmetry.

Our k-space log-magnitude plot visually confirms this centrosymmetry (it would look the same if rotated 180 degrees). Because the measured k-space data has this property, the inverse Fourier transform (our reconstruction) results in an image that is real-valued, with all the meaningful information captured in the real component.

C)

```

In [15]: import numpy as np
import matplotlib.pyplot as plt

filename = 'sepi.npy'

# 1. Load the data
raw_data = np.load(filename)

# 2. Reshape the data
# Reshape the 1D vector of 4096 points into a 2D (64, 64) k-space matrix
kspace = raw_data.reshape((64, 64))

```

```

# 3. Correct for EPI zig-zag
# We apply the correction to a copy to be safe
kspace_corrected = kspace.copy()

# Select every other row (1, 3, 5, ...)
# Reverse the order of elements in that row (::-1)
# Take the complex conjugate
kspace_corrected[1::2] = np.conjugate(kspace_corrected[1::2, ::-1])

# 4. Reconstruct the image
# Standard MR reconstruction pipeline:
# 1. Move k-space center to corner for FFT
# 2. 2D Inverse FFT
# 3. Move image center from corner back to center
img_complex = np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(kspace_corrected)))

# 5. Visualize the magnitude
img_magnitude = np.abs(img_complex)

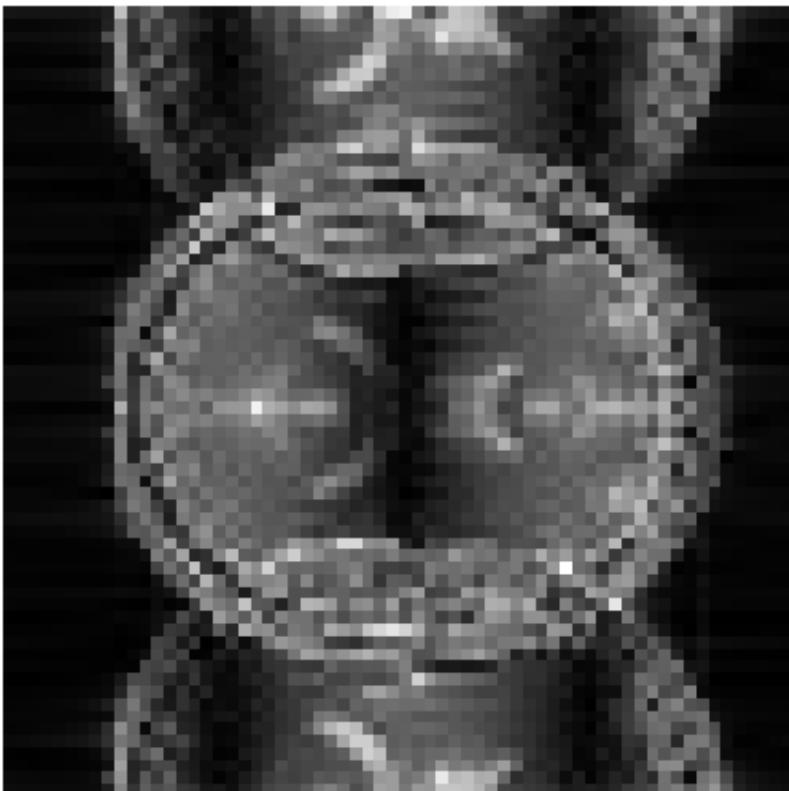
plt.imshow(img_magnitude, cmap='gray')
plt.title('Reconstructed EPI Image (Zig-Zag Corrected)')
plt.axis('off')

# Save the image
output_filename = 'epi_reconstruction_zigzag.png'
plt.savefig(output_filename)
print(f'Reconstructed image saved as {output_filename}')

```

Reconstructed image saved as epi_reconstruction_zigzag.png

Reconstructed EPI Image (Zig-Zag Corrected)



d)

In [16]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import shift

def find_subpixel_peak_shift(spc_file):
    print("--- Part 1: Finding Sub-pixel Shift ---")

    # --- 1. Load and process echo data ---
    spc_data = np.load(spc_file)
    # First 64 points are the positive echo
    echo_pos = spc_data[:64]
    # Second 64 points are the negative echo, which is time-reversed
    echo_neg_rev = spc_data[64:][::-1]

    # --- 2. Find sub-pixel peak locations ---
    def get_peak(data):
        """Finds sub-pixel peak by fitting a parabola to the max point."""
        peak_idx = np.argmax(np.abs(data))
        # Get 3 points around the peak for the fit
        y_points = np.abs(data[peak_idx-1 : peak_idx+2])
        # Fit y = ax^2 + bx + c to x = [-1, 0, 1]
        poly = np.polyfit([-1, 0, 1], y_points, 2)
        # The vertex (max) of the parabola is at x = -b / (2a)
        offset = -poly[1] / (2 * poly[0])
        return peak_idx + offset

    peak_pos = get_peak(echo_pos)
    peak_neg = get_peak(echo_neg_rev)

    # The shift to apply to the negative echo to align it with the positive
    pixel_shift = peak_pos - peak_neg

    print(f"Positive echo peak: {peak_pos:.4f}")
    print(f"Negative echo peak: {peak_neg:.4f}")
    print(f"Calculated shift: {pixel_shift:.4f} samples")

    # --- 3. Plot the echoes ---
    plt.figure(figsize=(10, 4))
    plt.title('Echo Peak Comparison')
    plt.plot(np.abs(echo_pos), 'b-o', label='Positive Echo', markersize=4)
    plt.plot(np.abs(echo_neg_rev), 'r--x', label='Negative Echo (Reversed)')
    # Plot vertical lines at calculated peaks
    plt.axvline(peak_pos, color='b', linestyle=':', label=f'Pos Peak ({peak_pos:.4f})')
    plt.axvline(peak_neg, color='r', linestyle=':', label=f'Neg Peak ({peak_neg:.4f})')
    plt.legend()
    plt.show()

    return pixel_shift

def correct_sepi_image(sepi_file, pixel_shift):
    print("\n Part 2: Applying Correction to Image")

    #1. Load and apply initial zig-zag correction
    kspace = np.load(sepi_file).reshape((64, 64))

    #Create a copy for the uncorrected "before" image
    kspace_zigzag_corr = kspace.copy()
```

```

kspace_zigzag_corr[1::2] = np.conjugate(kspace_zigzag_corr[1::2, ::-1])

#Create a second copy for the "after" image
kspace_shifted = kspace_zigzag_corr.copy()

#2. Apply sub-pixel shift correction
# We apply the calculated shift to the odd-numbered lines (the negative)
# using cubic interpolation (order=3).
print(f"Applying {pixel_shift:.4f} sample shift to odd k-space lines.")
for i in range(1, 64, 2):
    # We shift the complex data. scipy's shift handles this.
    kspace_shifted[i, :] = shift(kspace_shifted[i, :],
                                shift=pixel_shift,
                                order=3,
                                mode='constant',
                                cval=0.0)

#3. Apply phase correction
# We force the k-space center (DC component) to be real.
# This removes a global, arbitrary phase offset.
# The center of our 64x64 *centered* k-space is at (32, 32).
center_phase = np.angle(kspace_shifted[32, 32])
print(f"Applying global phase correction of {-center_phase:.4f} radians")
kspace_final = kspace_shifted * np.exp(-1j * center_phase)

#4. Reconstruct both images for comparison
# "Before" image (only zig-zag)
img_before = np.abs(np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(ksp

# "After" image (full correction)
img_after = np.abs(np.fft.fftshift(np.fft.ifft2(np.fft.ifftshift(ksp

#5. Display results
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.imshow(img_before, cmap='gray')
ax1.set_title('Figure 3 (Left) – Zig-Zag Correction Only')
ax1.axis('off')

ax2.imshow(img_after, cmap='gray')
ax2.set_title('Figure 3 (Right) – Full Correction')
ax2.axis('off')

plt.tight_layout()
plt.show()

#Main execution
# Part 1: Find the shift
shift_to_apply = find_subpixel_peak_shift('spc.npy')

# Part 2: Apply the shift
correct_epi_image('sepi.npy', shift_to_apply)

```

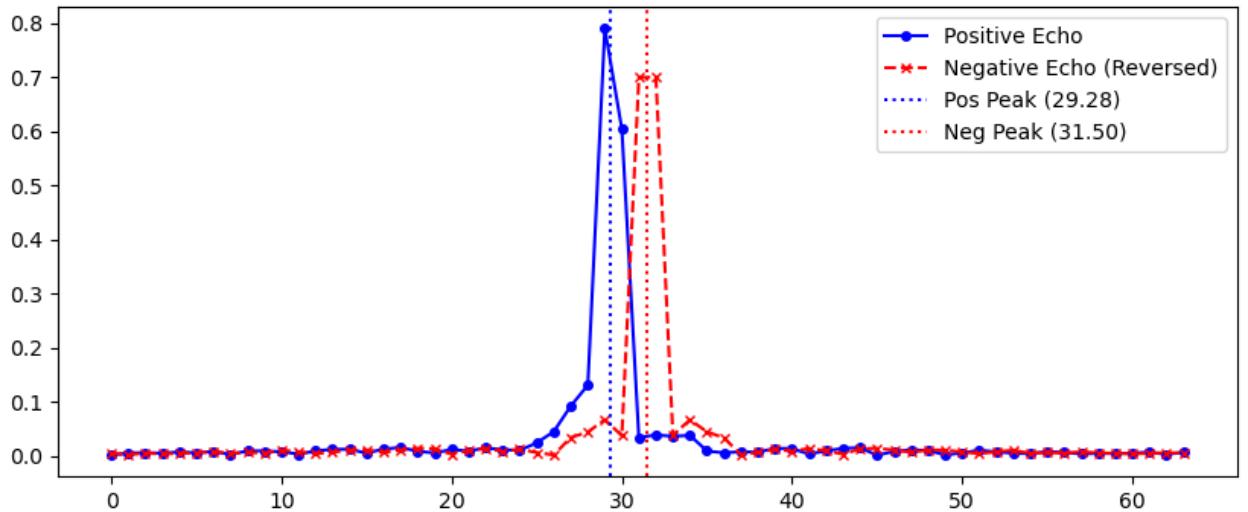
--- Part 1: Finding Sub-pixel Shift ---

Positive echo peak: 29.2804

Negative echo peak: 31.5014

Calculated shift: -2.2210 samples

Echo Peak Comparison



Part 2: Applying Correction to Image

Applying -2.2210 sample shift to odd k-space lines...

Applying global phase correction of 2.8420 radians...

Figure 3 (Left) - Zig-Zag Correction Only

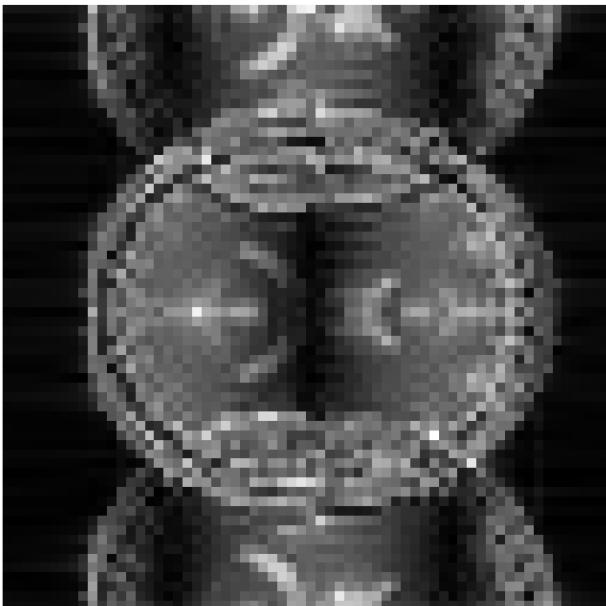


Figure 3 (Right) - Full Correction

