MRI image Reconstruction from K-space

Author: Livingstone Eli Ayivor

MRI images are not directly acquired; instead, data is collected in *k-space* (frequency domain).

- To reconstruct an image, we apply an inverse Fourier transform.
- Misunderstanding k-space often leads to reconstruction errors (e.g., misplaced frequencies, noise, artifacts).

This notebook shows a step-by-step approach to reconstructing an MRI image from K-space data. we'll make use of the M4Raw_kspace dataset from kaggle.

Install and Load Required Libraries

```
!pip -qq install fastmri
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.4/101.4 kB 3.2 MB/s eta
0:00:00
etadata (setup.py) ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
58.1/58.1 kB 2.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 1.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 27.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 11.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 7.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 66.0 MB/s eta
0:00:00
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
pylibcugraph-cu12 24.12.0 requires pylibraft-cu12==24.12.*, but you
have pylibraft-cu12 25.2.0 which is incompatible.
pylibcugraph-cu12 24.12.0 requires rmm-cu12==24.12.*, but you have
rmm-cu12 25.2.0 which is incompatible.
```

```
import h5py
import numpy as np
from matplotlib import pyplot as plt
```

```
import os
import fastmri
from fastmri.data import transforms as T
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation
```

Load sample data

```
file_path =
'/kaggle/input/m4raw-kspace/multicoil_train/2022061001_FLAIR01.h5'
hf = h5py.File(file_path)

volume_kspace = hf['kspace'][()]

middle_index = volume_kspace.shape[0] //2
slice_kspace = volume_kspace[middle_index] # Choosing the middle slice
of this volume
```

In multi-coil MRIs, k-space has the following shape: (number of slices, number of coils, height, width)

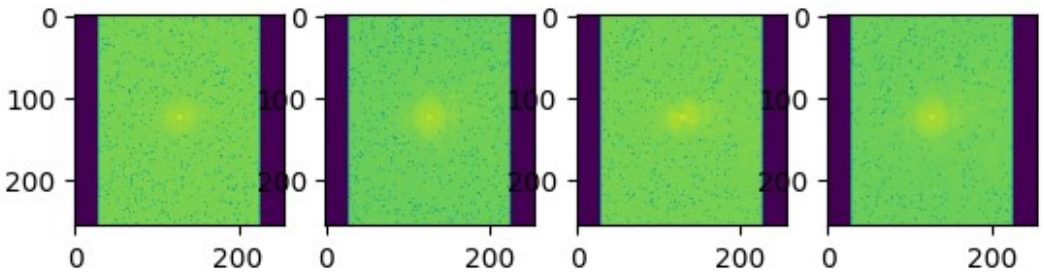
```
volume_kspace = hf['kspace'][()]
print('Kspace shape:',volume_kspace.shape)
out = hf['reconstruction_rss'][()]
print('reconstruction rss shape:',out.shape)
```

```
Kspace shape: (18, 4, 256, 256)
reconstruction rss shape: (18, 256, 256)
```

Let's see what the absolute value of k-space looks like

```
def show_coils(data, slice_nums, cmap=None):
    fig = plt.figure()
    for i, num in enumerate(slice_nums):
        plt.subplot(1, len(slice_nums), i + 1)
        plt.imshow(data[num], cmap=cmap)

show_coils(np.log(np.abs(slice_kspace) + 1e-9), [0, 1, 2, 3])  # This
shows coils 0, 1, 2 and 3
```



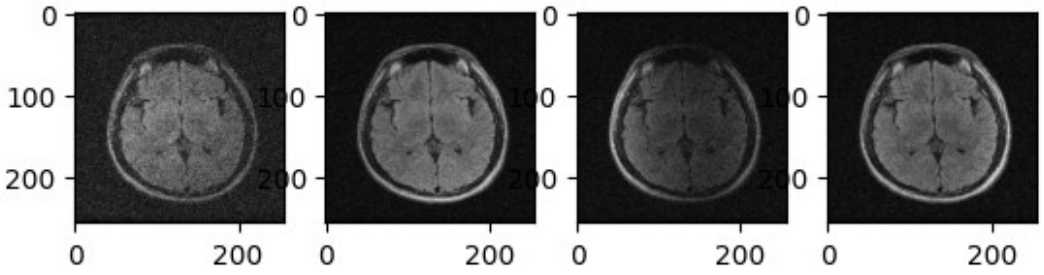
The glow in the centre of the k-space image represents the lower spatial frequencies, which contains the bulk of the image contrast information. As we move away from this central glow, the finer details and edges, representing the higher spatial frequencies, become apparent. These outer regions may appear fainter, but they hold vital information that sharpens our final image. In essence the outer region carries the fine details and edges of the image.

How do we get to an actual image?

To get from k-space to an actual image, we use the inverse Fourier transform. This process translates the frequency data from k-space into spatial information, giving us a clear image. When this transformation is applied to our k-space data, we're left with an image that reveals the underlying structures and details.

```
slice_kspace2 = T.to_tensor(slice_kspace)      # Convert from numpy
array to pytorch tensor
slice_image = fastmri.ifft2c(slice_kspace2)           # Apply Inverse
Fourier Transform to get the complex image
slice_image_abs = fastmri.complex_abs(slice_image)   # Compute
absolute value to get a real image

show_coils(slice_image_abs, [0, 1, 2, 3], cmap='gray') # This shows
coils 0, 1, 2 and 3
```



After applying the inverse Fourier transform to our k-space data, what we get isn't just a regular image but a complex-valued image. This translates to each pixel having both a real and imaginary component.

While each pixel has both a real and imaginary component, they combine to form two vital attributes: magnitude and phase. The magnitude tells us about the pixel's brightness, and it's

what we usually associate with the familiar MRI image. The phase, on the other hand, captures angular information. While the real and imaginary parts might seem random on their own, their combined effect through magnitude and phase unveils the detailed structure in our MRI.

Combining coils into a unified image

In MRI imaging, multiple coils are often used to capture data, each producing its own image. As we can see, each coil in a multi-coil MRI scan focusses on a different region of the image. To unify these images into one comprehensive picture, we use the **Root Sum of Squares (RSS)** technique. By employing RSS, for each pixel, the values from each coil are squared, summed, and then the square root is taken. The result is a single, combined image that consolidates the information from all coils.

```
slice_image_rss = fastmri.rss(slice_image_abs, dim=0)

plt.imshow(np.abs(slice_image_rss.numpy()), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7b8bd0e0e990>
```