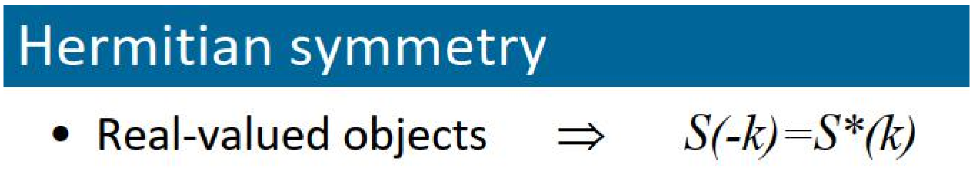
Computational MR imaging

Laboratory 3: Partial Fourier imaging

Nan Lan

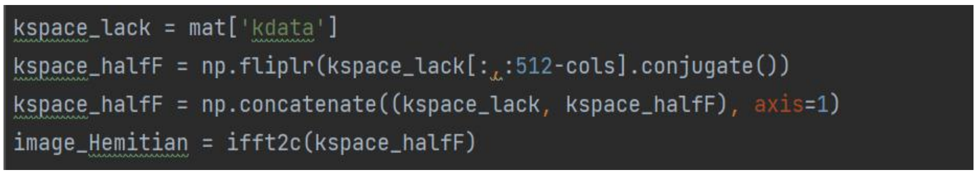
1. Hermitian symmetry reconstructed image

Use the Hermitian symmetry to estimate the zero_padding part of original kspace.



Hermitian symmetry

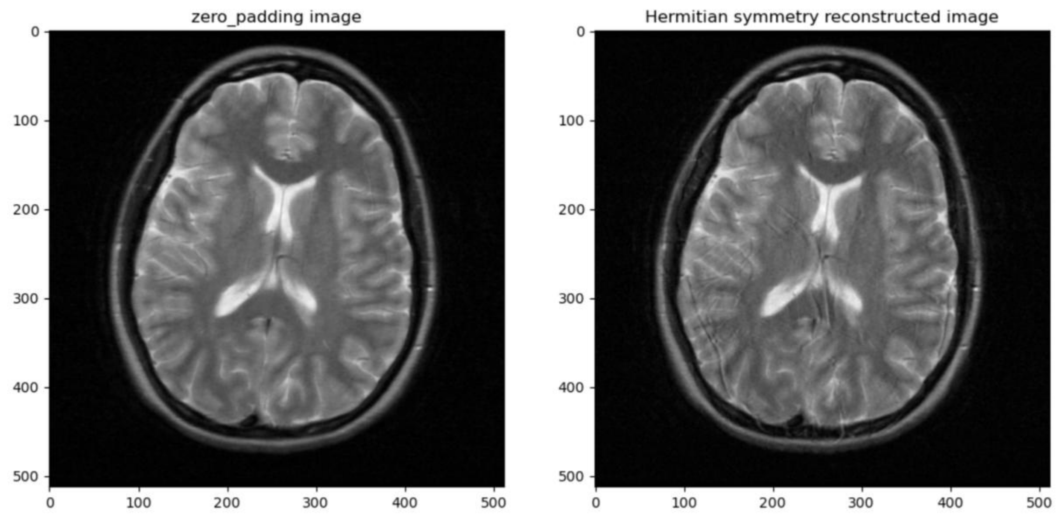
- Real-valued objects $\Rightarrow$ $S(-k)=S^*(k)$



```
kspace_lack = mat['kdata']
kspace_halfF = np.fliplr(kspace_lack[:,:512-cols].conjugate())
kspace_halfF = np.concatenate((kspace_lack, kspace_halfF), axis=1)
image_Hemitian = ifft2c(kspace_halfF)
```

The result below shows the PatialFourier=9/16 image and the Hermitian symmetry reconstructed image. The right image is less blurry than the left one. But Hermitian symmetry reconstructed image has obvious ringing artifact at the edge and in the center. The reason is that the condition of Hermitian symmetry is value in the image ought to be pure real, which means there should have no phase errors.

In fact, all image data sets contain some phase errors, and therefore the conjugate symmetry approximations are not perfect. The sources of these phase errors include the usual "posibilities": Bo inhomogeneity, susceptibility effects, eddy currents, physiologic motion and so on.



2. Process of Phase estimation from center kspace

1) Crop the center kspace and zero pad

```
kspace_cen = kspace_asym[:, 256 - 32:256 + 32]  # 512×64
kspace_cen_pad = np.pad(kspace_cen, ((0, 0), (224, 224)), 'constant', constant_values=(0, 0))
```

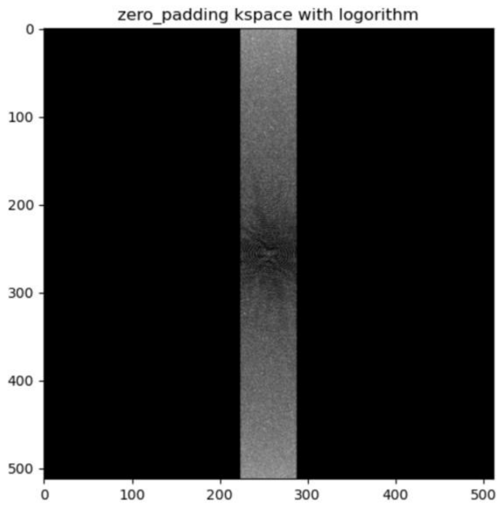
2) Create the hamming window and zero pad

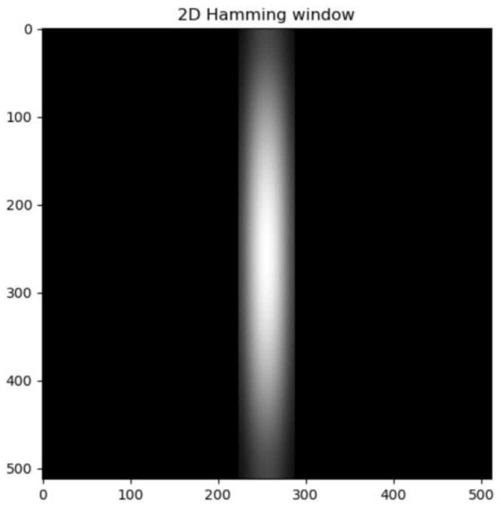
```
ham_win = np.sqrt(np.outer(hamming(512), hamming(64)))
ham_win_pad = np.pad(ham_win, ((0, 0), (224, 224)), 'constant', constant_values=(0, 0))
```

3) Use the filtered center space to estimate the phase

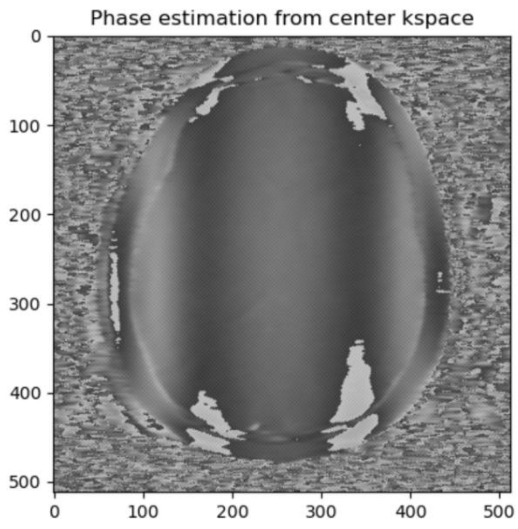
```
img_k_cen = ifft2c(kspace_cen_pad * ham_win_pad)
phase = np.angle(img_k_cen)
```

The result below shows the zaeo padding kspace with logirithm and the 2D hamming window.



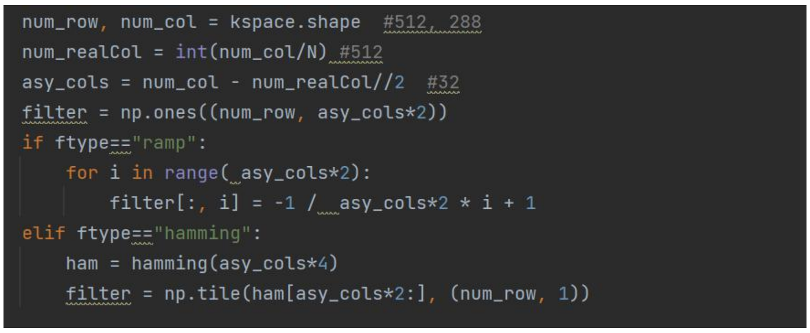


The result below is the phase estimation from the center kspace after filtering by 2D hamming window.



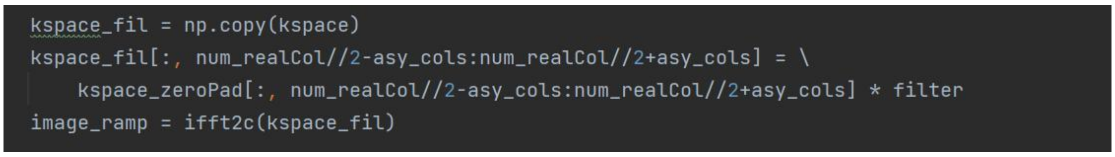
3. Process of Margosian method

1) Create ramp filter with the size of (512,32)



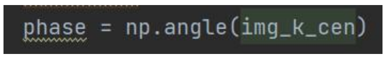
```
num_row, num_col = kspace.shape  #512, 288
num_realCol = int(num_col/N) #512
asy_cols = num_col - num_realCol//2  #32
filter = np.ones((num_row, asy_cols*2))
if ftype=="ramp":
    for i in range( asy_cols*2):
        filter[:, i] = -1 /  asy_cols*2 * i + 1
elif ftype=="hamming":
    ham = hamming(asy_cols*4)
    filter = np.tile(ham[asy_cols*2:], (num_row, 1))
```

2) Use ramp filter to smooth the transition between kspace and zero padding parts, so as to reduce gibbs ring artifact. Obtain I_0(r) now.

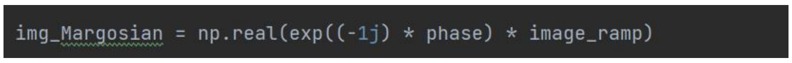


```
kspace_fil = np.copy(kspace)
kspace_fil[:, num_realCol//2-asy_cols:num_realCol//2+asy_cols] = \
    kspace_zeroPad[:, num_realCol//2-asy_cols:num_realCol//2+asy_cols] * filter
image_ramp = ifft2c(kspace_fil)
```

3) Reconstruct image with phase part and magnitude part based on Margosian method



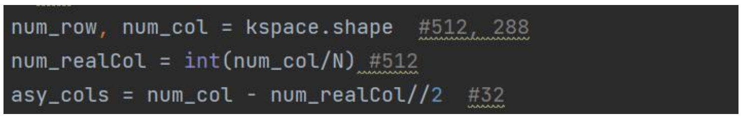
```
phase = np.angle(img_k_cen)
```



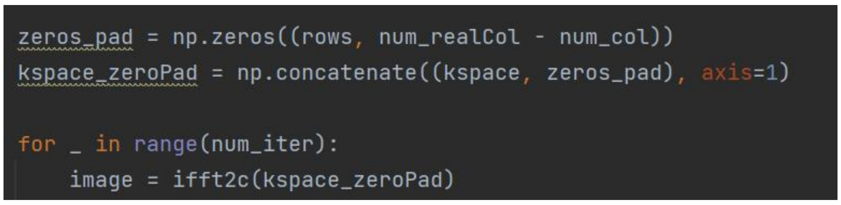
```
img_Margosian = np.real(exp((-1j) * phase) * image_ramp)
```

4. Process of POCS method

1) Initialize and use ifft to obtain the zero_padding image

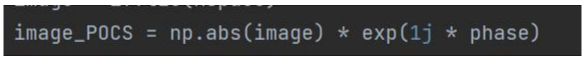


```
num_row, num_col = kspace.shape  #512, 288
num_realCol = int(num_col/N) #512
asy_cols = num_col - num_realCol//2  #32
```



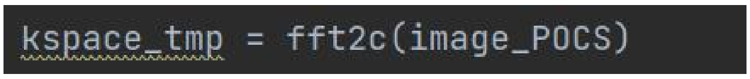
```
zeros_pad = np.zeros((rows, num_realCol - num_col))
kspace_zeroPad = np.concatenate((kspace, zeros_pad), axis=1)

for _ in range(num_iter):
    image = ifft2c(kspace_zeroPad)
```

2) Reconstruct image with phase part and magnitude of zero_padding image



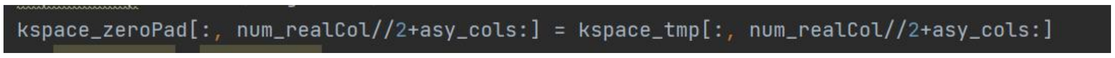
```
image_POCS = np.abs(image) * exp(1j * phase)
```

3) Use fft to obtain the new kspace



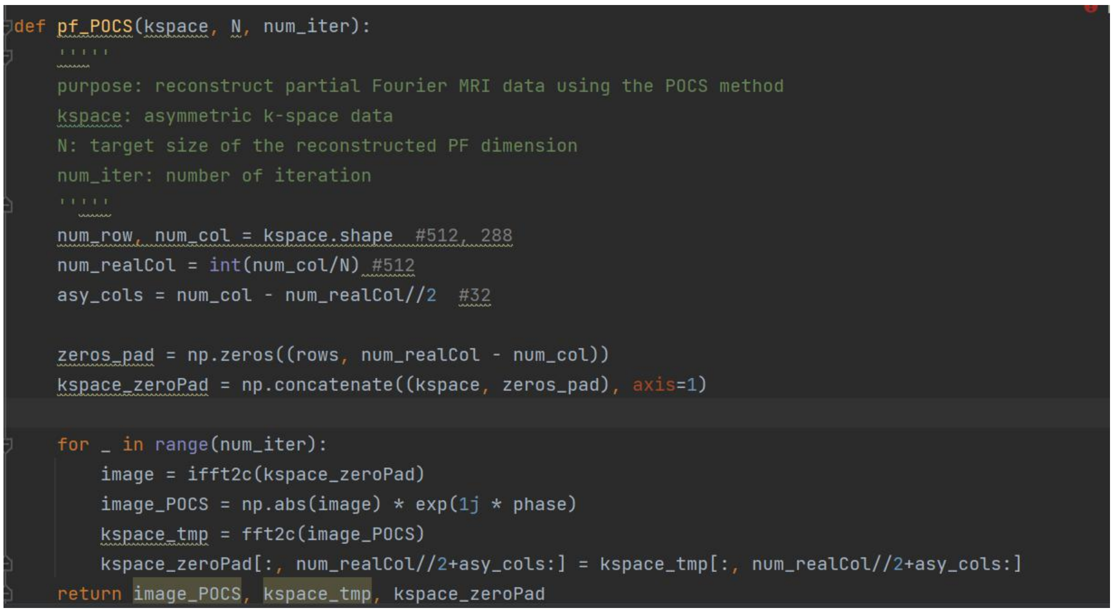
```
kspace_tmp = fft2c(image_POCS)
```

4) Fill the zero padding part in orignal kspace with the corresponding part of new kspace, and go to step1



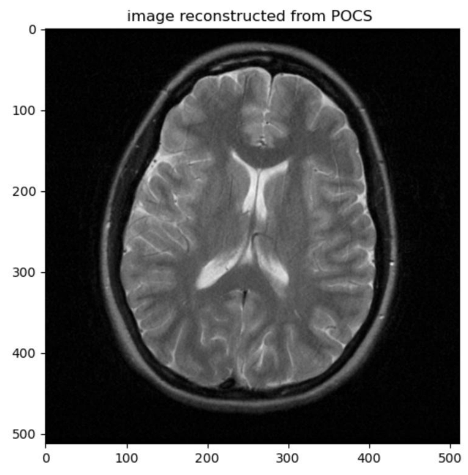
```
kspace_zeroPad[:, num_realCol//2+asy_cols:] = kspace_tmp[:, num_realCol//2+asy_cols:]
```

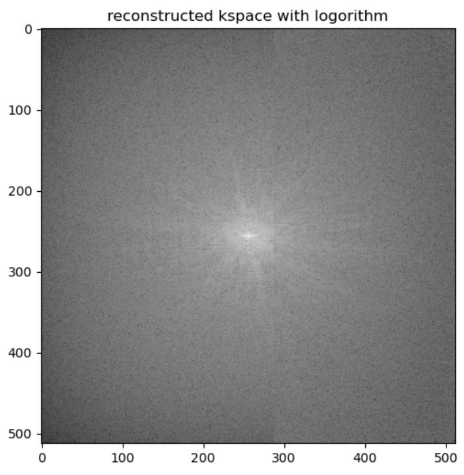
The code below is the whole process:

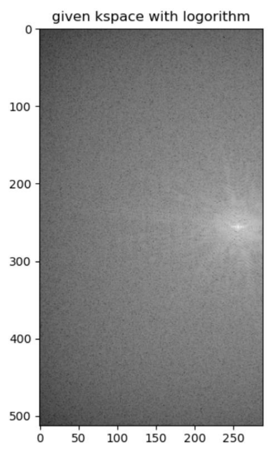


```
def pf_POCS(kspace, N, num_iter):
    '''''
    purpose: reconstruct partial Fourier MRI data using the POCS method
    kspace: asymmetric k-space data
    N: target size of the reconstructed PF dimension
    num_iter: number of iteration
    '''''
    num_row, num_col = kspace.shape  #512, 288
    num_realCol = int(num_col/N) #512
    asy_cols = num_col - num_realCol//2  #32

    zeros_pad = np.zeros((rows, num_realCol - num_col))
    kspace_zeroPad = np.concatenate((kspace, zeros_pad), axis=1)

    for _ in range(num_iter):
        image = ifft2c(kspace_zeroPad)
        image_POCS = np.abs(image) * exp(1j * phase)
        kspace_tmp = fft2c(image_POCS)
        kspace_zeroPad[:, num_realCol//2+asy_cols:] = kspace_tmp[:, num_realCol//2+asy_cols:]
    return image_POCS, kspace_tmp, kspace_zeroPad
```

The pictures below are the reconstructed image from POCS , the given kspace and the reconstructed kspace where the right part of given kspace is filled by the estimated kspace.

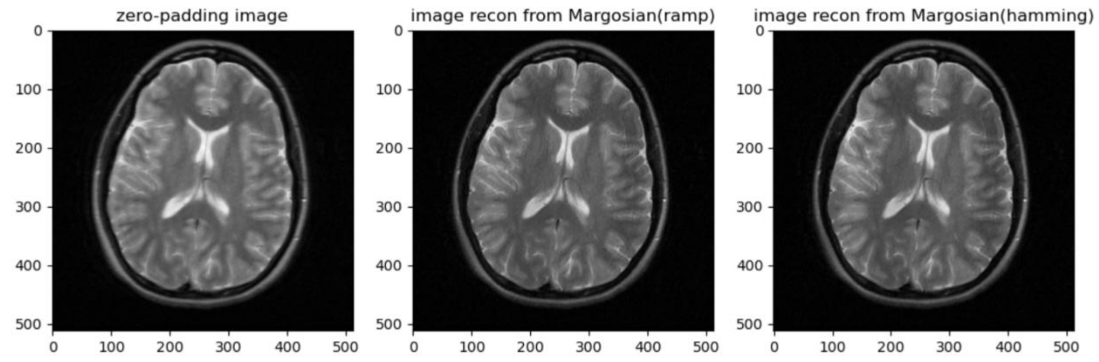






5. Result of Margosian method

The result below shows the result of margosian method.
Compared withe zero-padding image, the reconstructed image from Margosian has higher spatial resolution but also stronger gibbs ring.
There are no big different between Margosian(ramp) and Margosian(ramp)


zero-padding image
image recon from Margosian(ramp)
image recon from Margosian(hamming)

6. Different iteration of POCS Method

The result below shows Different iteration of POCS Method. The large the the iteration number is, the less blurry the reconstructed image is. When the iteration number reach 6, the the algorithm has converged, because the upcoming reconstructed image doesn't change much.

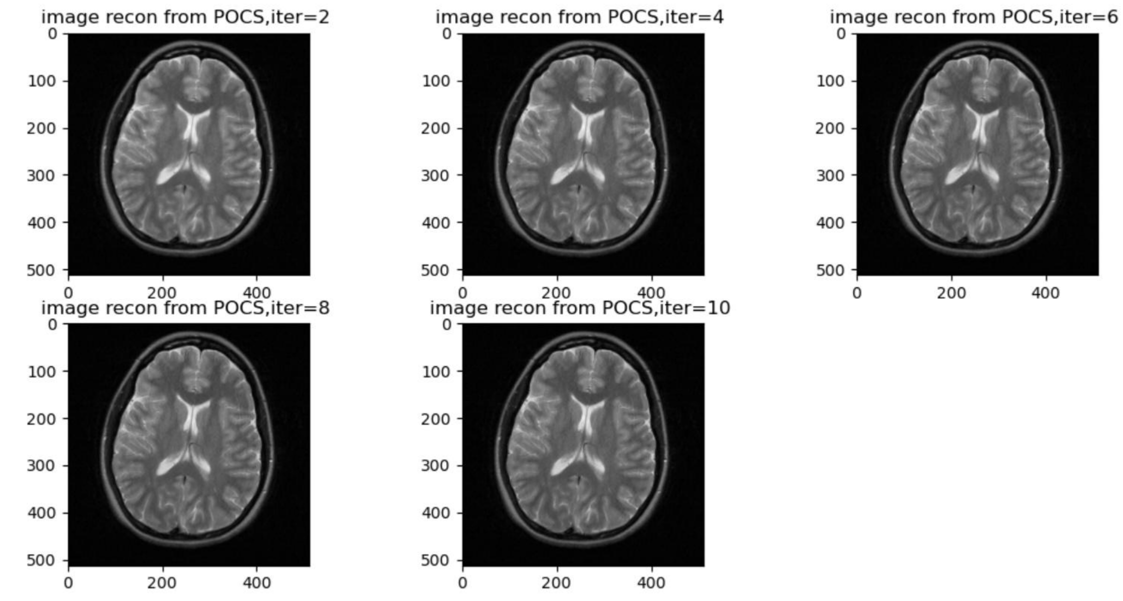
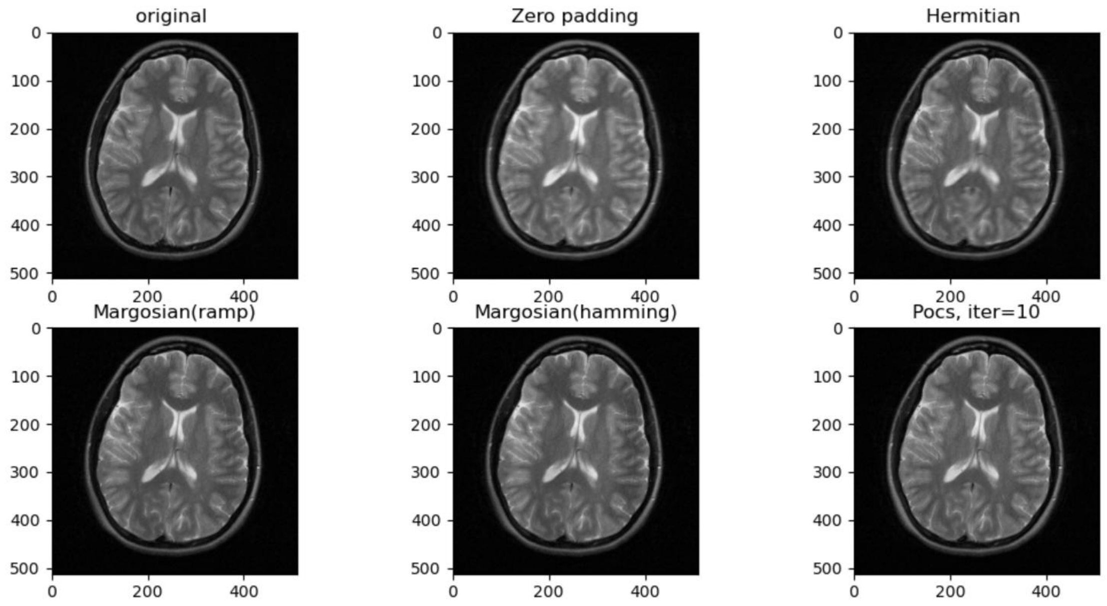


image recon from POCS,iter=2
image recon from POCS,iter=4
image recon from POCS,iter=6
image recon from POCS,iter=8
image recon from POCS,iter=10

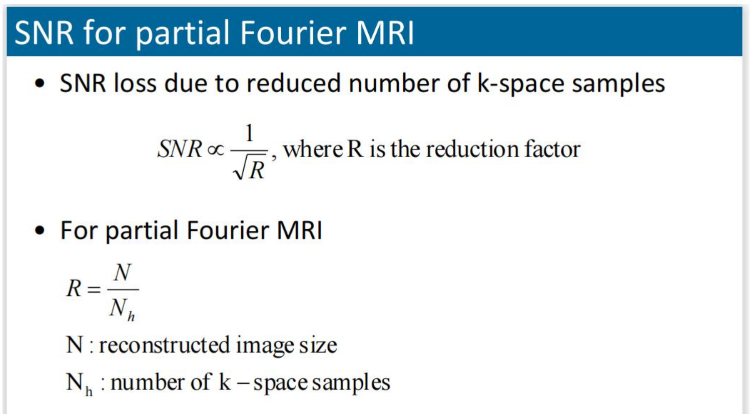
7. Hermitian, Margosian, POCS methods comparisom

The result below shows the reconstruted image from Original, Hermitian, Margosian(ramp), Margosian(hamming), POCS methods.



Now we compare these different methods in terms of SNR, spatial resolution, residual artifacts and ringing

SNR: The POCS and Margosian methods have lower SNR. Take the definition of SNR for partial Fourier MRI (refer to the image below). R = 16/9 in our example is larger than 1, so 1/sqrt(R) would be less than 1.



SNR for partial Fourier MRI

- SNR loss due to reduced number of k-space samples

$$SNR \propto \frac{1}{\sqrt{R}}, \text{ where R is the reduction factor}$$

- For partial Fourier MRI

$$R = \frac{N}{N_h}$$

N : reconstructed image size

N_h : number of k – space samples

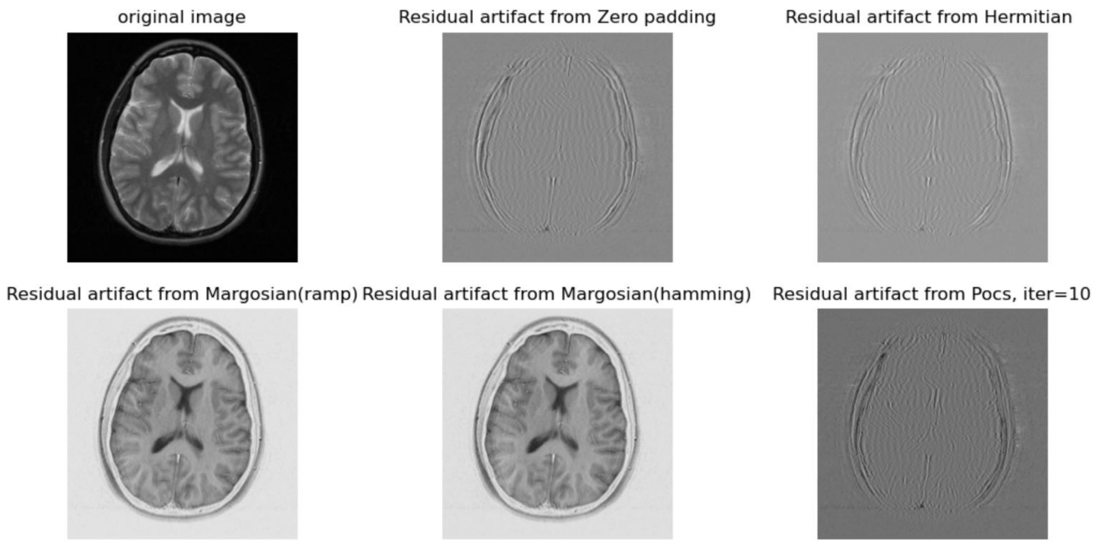
Source: computer MRI slice

Spacial resolution: The POCS and Margosian methods can get the better reconstructed image than Hermitian method. POCS method has higher spacial resolution than Margosian method, because it estimate the not only the phase but the zero part of kspace.

Residual Artifact: The result below shows the residual artifact which is calculated by: the magnitude of reconstruted image minus the magnitude of original image.

POCS method can help to reduce the residual artifact.

Residual artifact from margosian looks like a image, because the image reconstructed by the algorithm of margosian has only real part but the original image has both real and imaginary parts. Actually the residual artifact from margosian is not supposed to be likw this.


original image
Residual artifact from Zero padding
Residual artifact from Hermitian
Residual artifact from Margosian(ramp)
Residual artifact from Margosian(hamming)
Residual artifact from Pocs, iter=10

Ringing: Hermisian will lead to stronger gibbs ringing than the POCS method and Margosian . The reason why Hermisain has stronger gibbs ringing is that the condition of Hermitian symmetry is value in the image ought to be pure real, which means there should have no phase errors. For detailed explaination, please refer to question2.