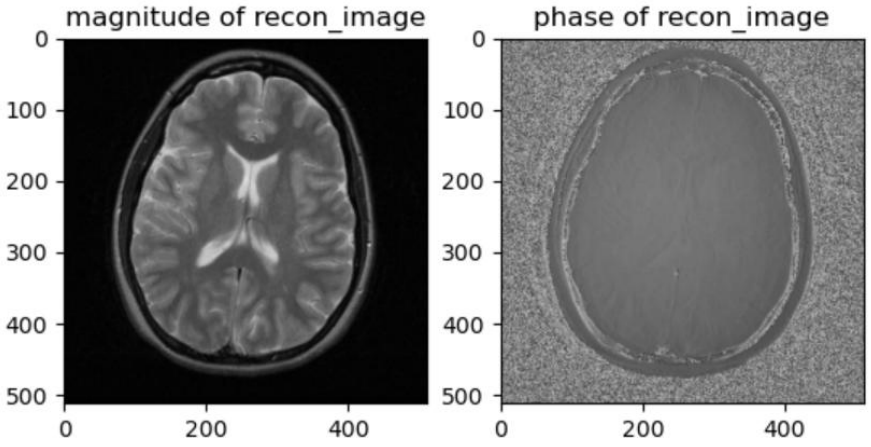
Computational MR imaging

Laboratory 2: k-space sampling and Fourier reconstruction

Nan Lan

1. FFT reconstruction of Cartesian MRI data

The image below is the magnitude and phase of the reconstructed image



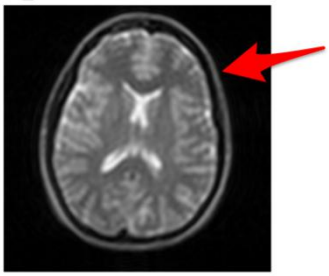

```python
# load matlab file
mat = scipy.io.loadmat('kdata1.mat')
kspace = mat['kdata1']

###### 1. FFT reconstruction of Cartesian MRI data
def fft2c(img):
    kspace = np.fft.fft2(img)
    kspace = np.fft.fftshift(kspace)
    return kspace

def ifft2c(kspace):
    image = np.fft.ifft2(kspace)
    img = np.fft.ifftshift(image)
    return img

recon_image = ifft2c(kspace)
magnitube = np.abs(recon_image)
angle = np.angle(recon_image,deg=True)
```

2. Effects of k-space zero-padding and zero padding

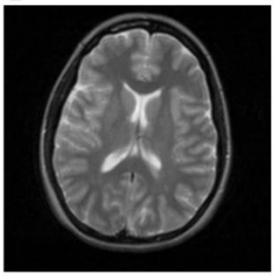
The result below shows the the reconstructed images. The center of kspace is the low frequency region, which contain the general overview and outline of image. The boundary of kspace is the low frequency region, which represents the edge and detailed information of MRI image. When the kspace is truncated and padded, the edge and detailed information lost. The more the kspace is truncated, the more blurry the MRI image.

Besides, Gibbs ringing occurs at the boundary, especially in image recontruced by heavily truncated kspace, due to the truncation.

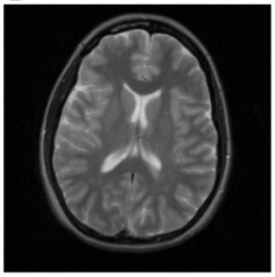
recon_image from 64×64 kspace



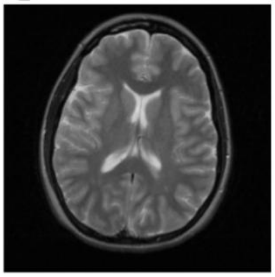
recon_image from 128×128 kspace



recon_image from 256×256 kspace



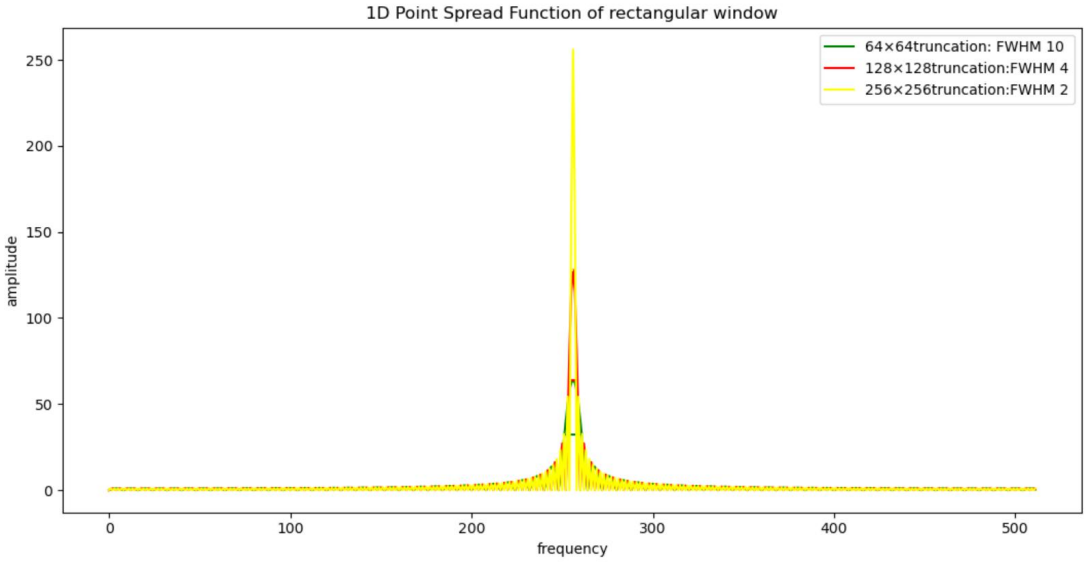
recon_image from original kspace



```python
###### 2. Effects of k-space zero-padding and zero padding
def get_truncated_kspace(kspace,width):
    half_w = width//2
    kspace = kspace[256 - half_w:256 + half_w, 256 - half_w:256 + half_w]
    return kspace

def zero_padding(kspace):
    pad_num = 256-len(kspace)//2
    zero_pad_kspace = np.pad(kspace, ((pad_num, pad_num), (pad_num, pad_num)), 'constant', constant_values=0)
    return zero_pad_kspace

kspace_64 = get_truncated_kspace(kspace, 64)
kspace_64_pad = zero_padding(kspace_64)
kspace_128 = get_truncated_kspace(kspace, 64)
kspace_128_pad = zero_padding(kspace_128)
kspace_256 = get_truncated_kspace(kspace, 64)
kspace_256_pad = zero_padding(kspace_256)

recon_image_64 = ifft2c(kspace_64_pad)
recon_image_128 = ifft2c(kspace_128_pad)
recon_image_256 = ifft2c(kspace_256_pad)
```

3. Point spread function (PSF)

After the Fourier transform, the spectral peaks generated by the voxel spread to its neighbors in the form of ripples, which is mathematically described as a point spread function (psf). Full-width at half-maximum (FWHM) can describe the blurry degree. PSF function leads to information leakage and information aliasing. The larger the FWHM, the more blurred the image.
The result below shows PSF functions of rectangular window with different FWHM.

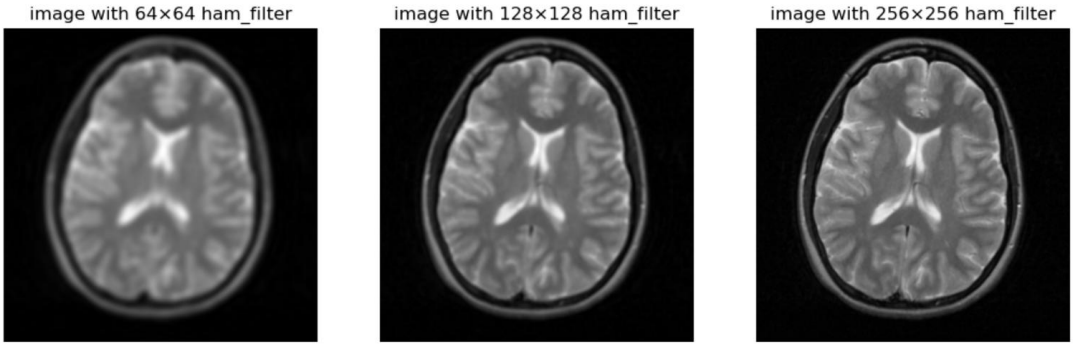



```
###### 3. Point spread function (PSF)
def get_PSF_fre(width):
    cols = len(kspace[0])
    psf = np.zeros((cols))
    psf[cols // 2 -width//2 : cols // 2 + width//2] = 1
    psf_fre = np.fft.fftshift(np.fft.fft(psf))
    return psf_fre

def get_FWHM(psf_fre):
    half_maximum = max(np.abs(psf_fre))//2
    new_fun = np.abs(psf_fre)-half_maximum
    left = np.argmin(np.abs(new_fun[:len(psf_fre)//2]))
    FWHM = (len(psf_fre)//2-left)*2
    return left, half_maximum, FWHM

psf_fre2= get_PSF_fre(64)
left2, half_maximum2, FWHM2 = get_FWHM(psf_fre2)
psf_fre6 = get_PSF_fre(128)
left6, half_maximum6, FWHM6 = get_FWHM(psf_fre6)
psf_fre10 = get_PSF_fre(256)
left10, half_maximum10, FWHM10 = get_FWHM(psf_fre10)
```

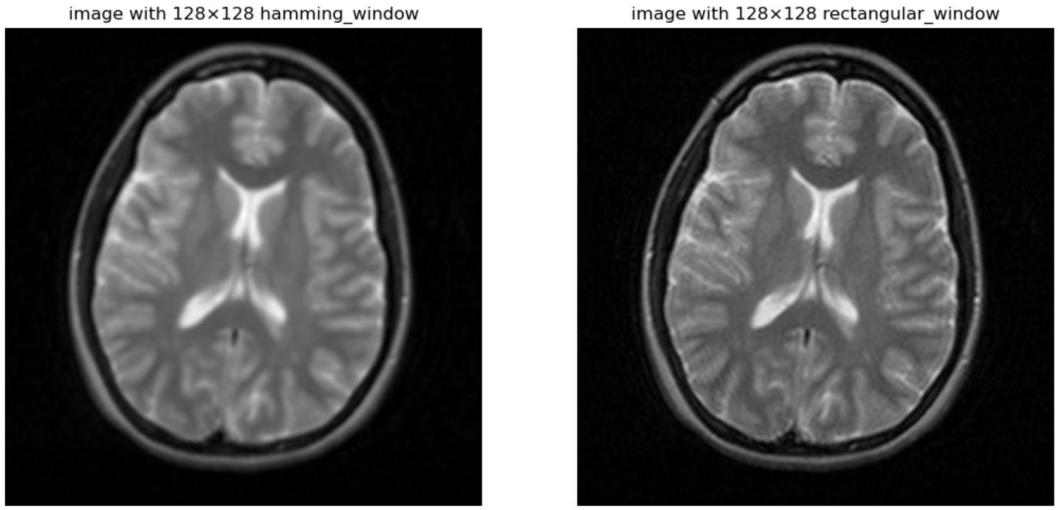
4. k-space filtering (windowing)

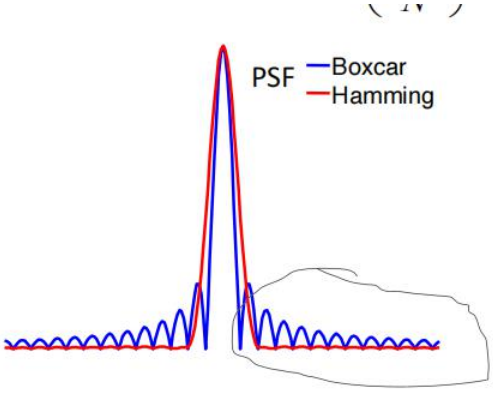
4.1 Image with haming window

The image below shows the reconstruced image from different truncated kspace and with respective hamming window.



From the result below, we can compare the effect of hamming filter. The image with hamming filter are more blurry but with less gibbs ring. When hamming filter is used, passby ripples are filtered out, which make the image smoother or on the other word blurry. But meanwhile it reduced the impact of gibbs ring, which is also caused by the ripples. This is a tradeoff between blur and gibbs ring.





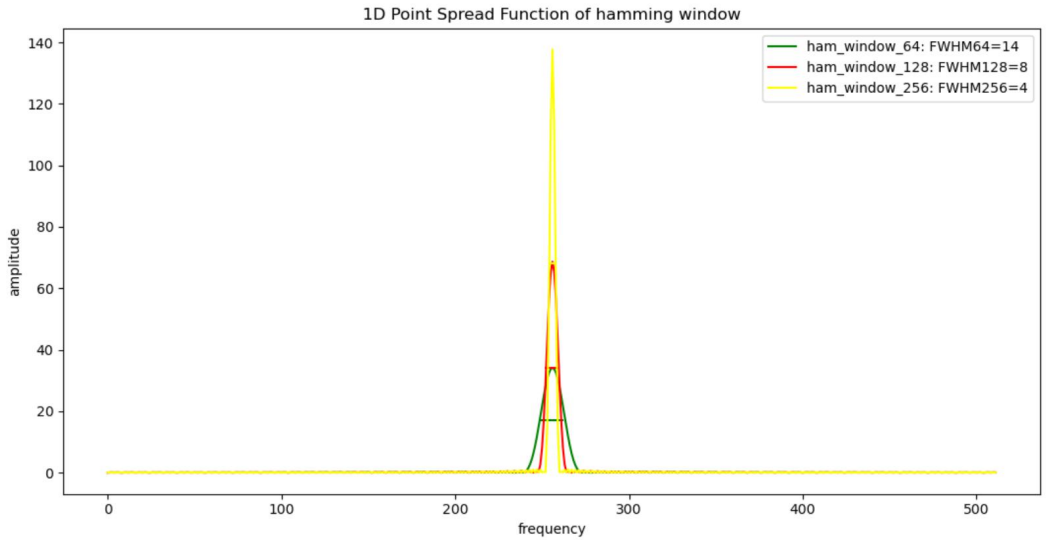
```
###### 4. k-space filtering (windowing)
##4.1 Image with haming window
def get_hamming_img(kspace, win_sz):
    window_1d = signal.hamming(win_sz)
    window_2d = np.sqrt(np.outer(window_1d, window_1d))
    window_pad_2d = zero_padding(window_2d)
    ham_img = ifft2c(window_pad_2d * kspace)
    return ham_img, window_pad_2d

ham_img_64, _ = get_hamming_img(kspace_64_pad, 64)
ham_img_128, _ = get_hamming_img(kspace_128_pad, 128)
ham_img_256, _ = get_hamming_img(kspace_256_pad, 256)
```

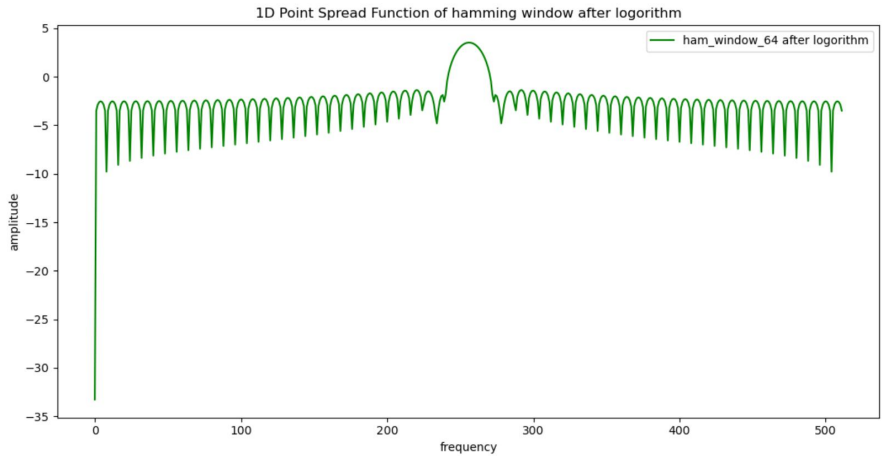
4.2 PSF of Hamming window

The result below shows the PSF of hamming window with different respective 64, 128,256 width. Full-width at half-maximum (FWHM) can describe the blurry degree. Compared with PSF of rectangular window, the FWHM of hamming windows are larger. From this point of view, we can also come to the conclusion that image with Hamming window are more blurry than rectangular window.

In summary, hamming window has lower resolution(more blurry) and less gibbs ring; rectangular window has high resolution and more gibbs ring

```
FWHM of rectangular window is respectively 10 4 2
FWHM of hamming window is respectively 14 8 4
```

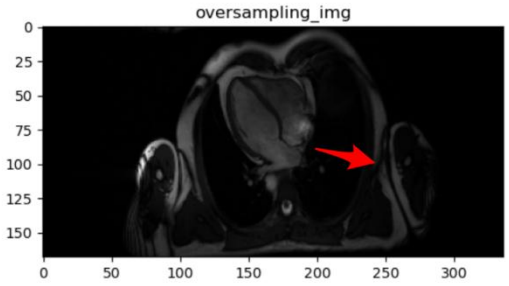


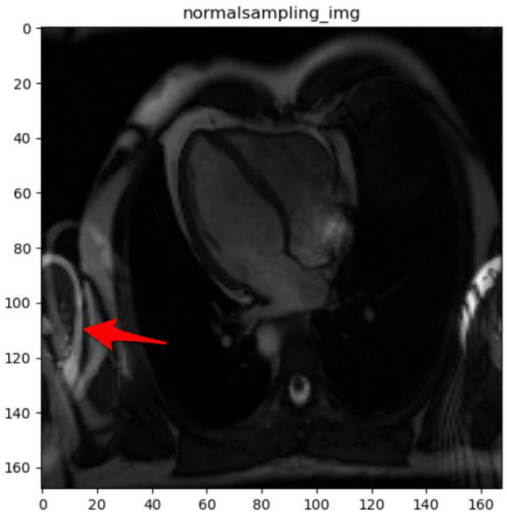



```python
##4.2 PSF of Hamming window
def get_ham_psf(ham_sz):
    half_sz = ham_sz//2
    window_64 = signal.hamming(ham_sz)
    window_64 = np.pad(window_64, (256 - half_sz, 256 - half_sz), constant_values=0)
    ham_window_64 = np.fft.fftshift(np.fft.fft(window_64))
    return ham_window_64

ham_window_64 = get_ham_psf(64)
ham_window_128 = get_ham_psf(128)
ham_window_256 = get_ham_psf(256)

left64, half_maximum64, FWHM64 = get_FWHM(ham_window_64)
left128, half_maximum128, FWHM128 = get_FWHM(ham_window_128)
left256, half_maximum256, FWHM256 = get_FWHM(ham_window_256)
```

5. Oversampling the readout dimension

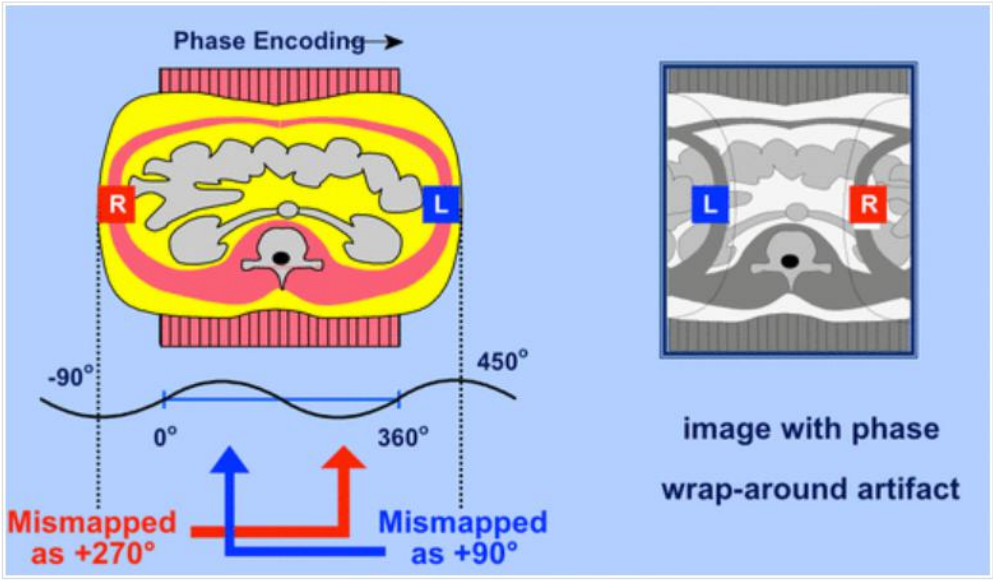
The left result is the image reconstructed by 2 times oversampling in read out dimension. The right result is the image, which doesn’t oversample. In the right result, phase wrap-around artifact occur, because the dimensions of an object exceed the defined field-of-view (FOV). Parts of the object extending beyond the field-of-view possess phases less than 0° or greater than phase dimension(in our result, phase dimension=168). The left side of the patient's body will therefore be "wrapped around" and spatially mismapped to the opposite (right) side of the image. A similar process will wrap the patient's right side around to the left.






```
###### 5. Oversampling the readout dimension
mat = scipy.io.loadmat('kdata2.mat')
kspace2 = mat['kdata2'] #168×336
oversampling_img = ifft2c(kspace2)
recon_kspace2 = kspace2[:,0::2] #168×168
normalsampling_img = ifft2c(recon_kspace2)
```

The image below explain the process of wrap-around artifact.



Origin of the wrap-around artifact