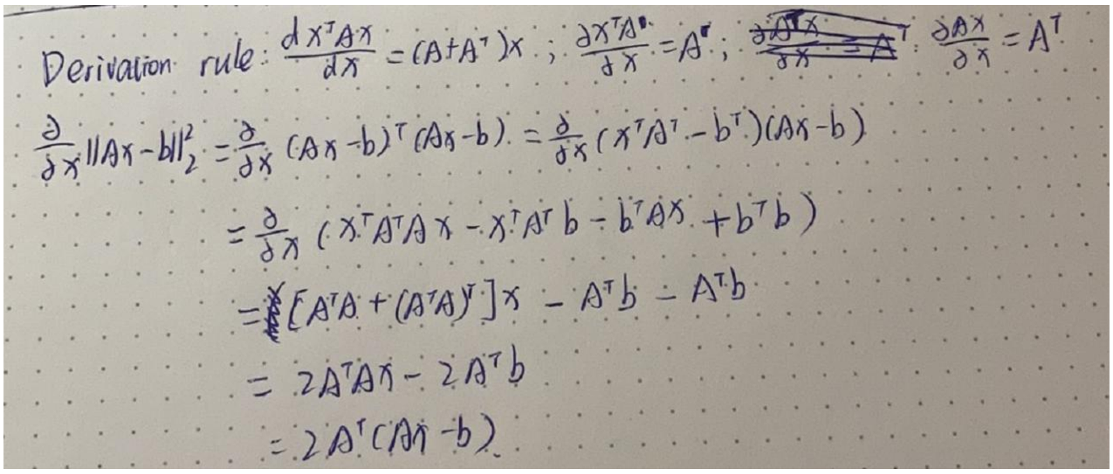
Computational MR imaging

Laboratory 7: Parallel Imaging III: Non-Cartesian Imaging and Iterative Reconstruction

Nan Lan

1. Derivation of gradient descent (analytical):



Derivation rule: $\frac{d x^T A x}{d x} = (A + A^T) x$; $\frac{\partial x^T A}{\partial x} = A$; $\cancel{\frac{\partial A^T x}{\partial x} = A^T}$ $\frac{\partial A x}{\partial x} = A^T$

$$\frac{\partial}{\partial x} \|Ax - b\|_2^2 = \frac{\partial}{\partial x} (Ax - b)^T (Ax - b) = \frac{\partial}{\partial x} (x^T A^T - b^T)(Ax - b)$$

$$= \frac{\partial}{\partial x} (x^T A^T A x - x^T A^T b - b^T A x + b^T b)$$

$$= [A^T A + (A^T A)^T] x - A^T b - A^T b$$

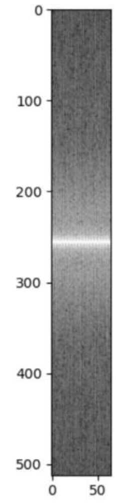
$$= 2A^T A x - 2A^T b$$

$$= 2A^T (Ax - b)$$

2. Iterative image reconstruction with gradient descent

2.1. Plot the kdata and at the sampling trajectory

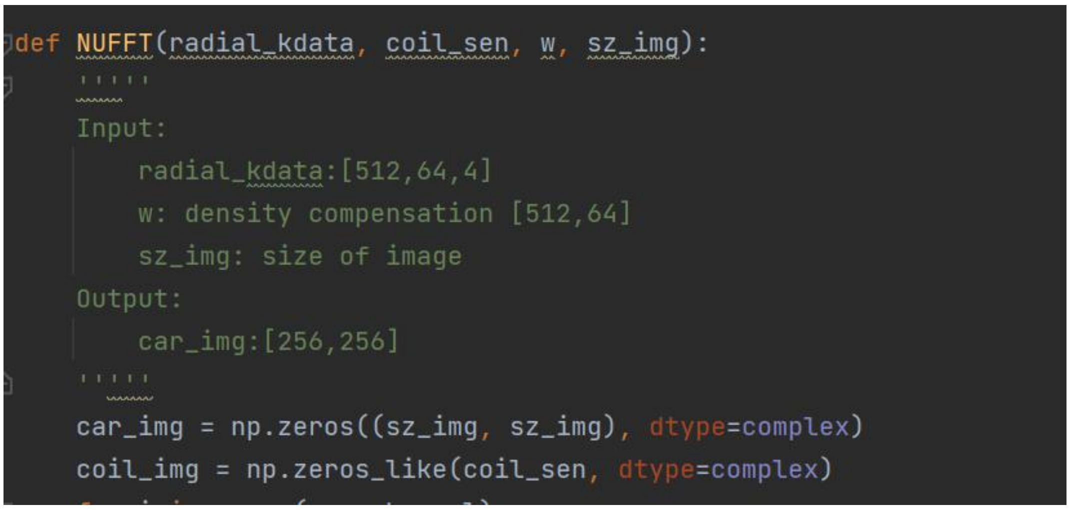
Each column corresponds to the readout dimension for each radial line.



2.2 NUFFT

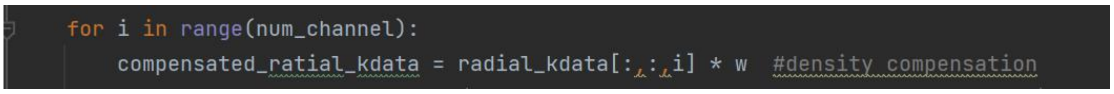
The process of NUFFT is as follow:

(1) Initialization of cartesian image and coil image;



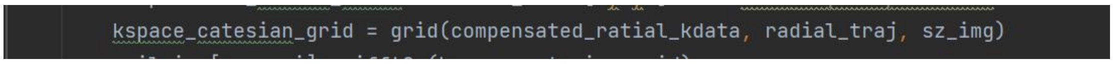
```
def NUFFT(radial_kdata, coil_sen, w, sz_img):
    '''''
    Input:
        radial_kdata:[512,64,4]
        w: density compensation [512,64]
        sz_img: size of image
    Output:
        car_img:[256,256]
    '''''
    car_img = np.zeros((sz_img, sz_img), dtype=complex)
    coil_img = np.zeros_like(coil_sen, dtype=complex)
```

(2) Density compensation of kdata;



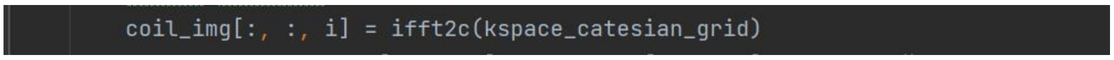
```
    for i in range(num_channel):
        compensated_ratial_kdata = radial_kdata[:,:,i] * w  #density compensation
```

(3) Mapping ratial kdata to cartesian grid;



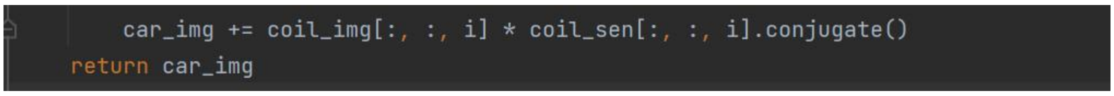
```
        kspace_catesian_grid = grid(compensated_ratial_kdata, radial_traj, sz_img)
```

(4) Inverse fourier transform of kspace_catesian_grid and get coil image;



```
        coil_img[:, :, i] = ifft2c(kspace_catesian_grid)
```

(5) Combine coil image and get cartesian image

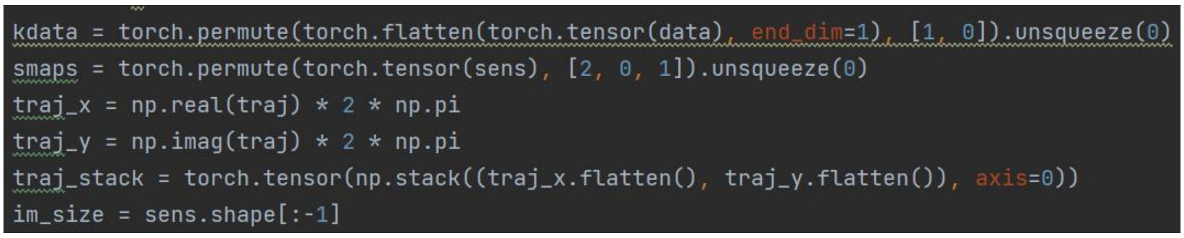


```
        car_img += coil_img[:, :, i] * coil_sen[:, :, i].conjugate()
    return car_img
```

2.3 Gradient Descent Reconstruction

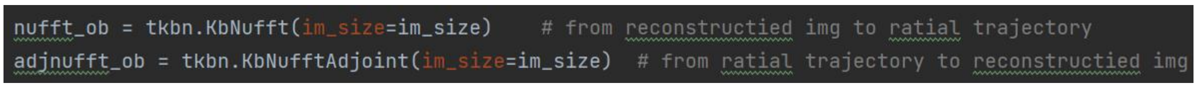
The process of gradient descent reconstrution is as follow:

(1) Construct kdata trajectory, which is suitatble for KbNufft toolbox.



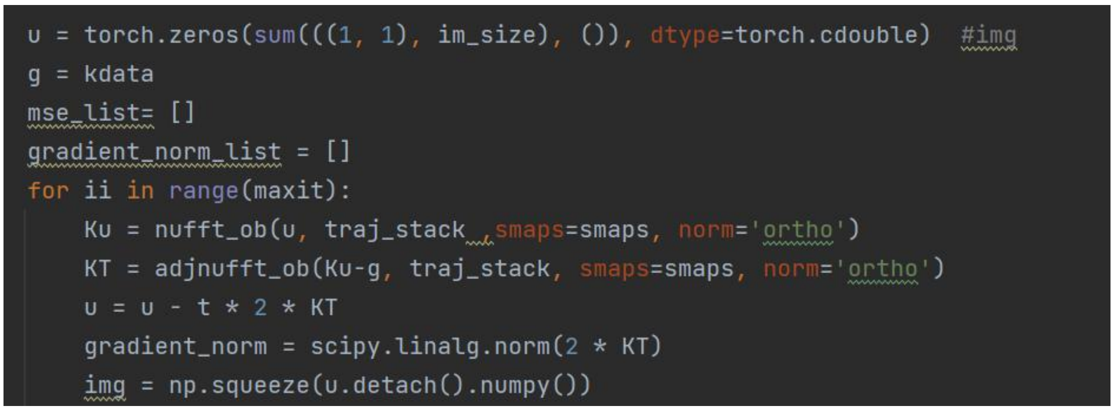
```
kdata = torch.permute(torch.flatten(torch.tensor(data), end_dim=1), [1, 0]).unsqueeze(0)
smaps = torch.permute(torch.tensor(sens), [2, 0, 1]).unsqueeze(0)
traj_x = np.real(traj) * 2 * np.pi
traj_y = np.imag(traj) * 2 * np.pi
traj_stack = torch.tensor(np.stack((traj_x.flatten(), traj_y.flatten()), axis=0))
im_size = sens.shape[:-1]
```

(2) Instantiate the mappings from cartesian image to radial trajectory and from radial trajectory to cartesian image.



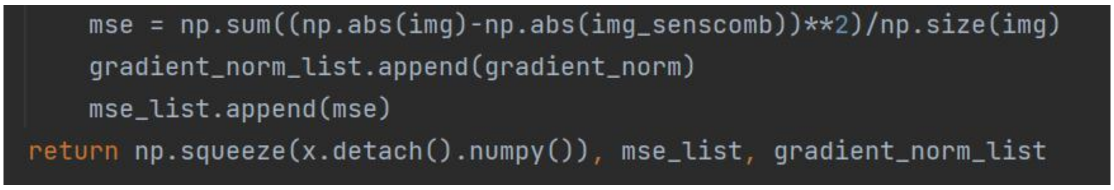
```
nufft_ob = tkbn.KbNufft(im_size=im_size)    # from reconstructied img to ratial trajectory
adjnufft_ob = tkbn.KbNufftAdjoint(im_size=im_size)  # from ratial trajectory to reconstructied img
```

(3) Run gradient descent method



```
u = torch.zeros(sum(((1, 1), im_size), ()), dtype=torch.cdouble)  #img
g = kdata
mse_list= []
gradient_norm_list = []
for ii in range(maxit):
    Ku = nufft_ob(u, traj_stack, smaps=smaps, norm='ortho')
    KT = adjnufft_ob(Ku-g, traj_stack, smaps=smaps, norm='ortho')
    u = u - t * 2 * KT
    gradient_norm = scipy.linalg.norm(2 * KT)
    img = np.squeeze(u.detach().numpy())
```

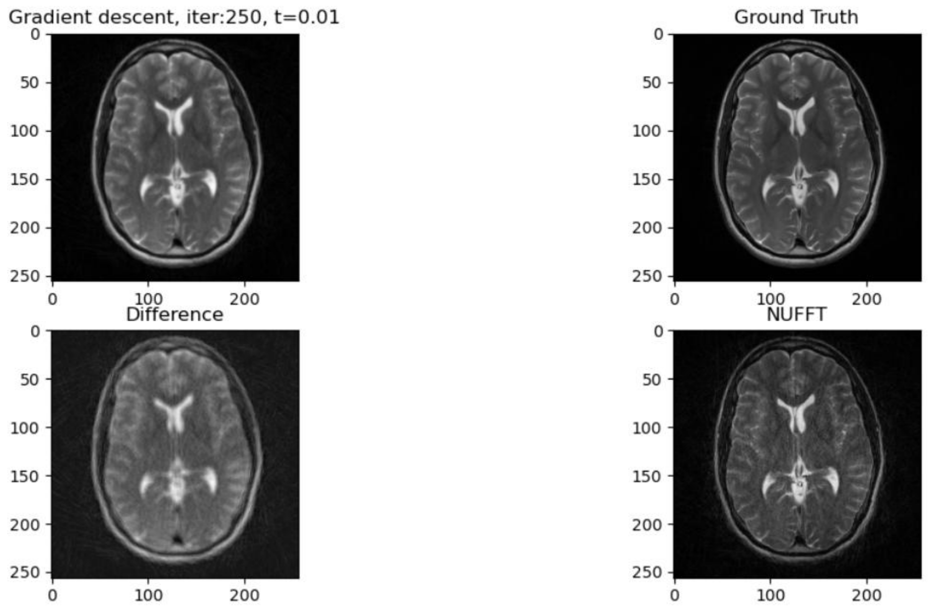
(4) Calculate mean square error and l2 norm of the gradient over the iterations



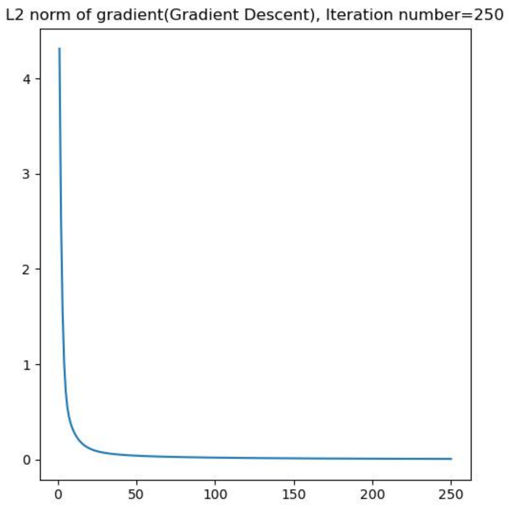
```
    mse = np.sum((np.abs(img)-np.abs(img_senscomb))**2)/np.size(img)
    gradient_norm_list.append(gradient_norm)
    mse_list.append(mse)
return np.squeeze(x.detach().numpy()), mse_list, gradient_norm_list
```

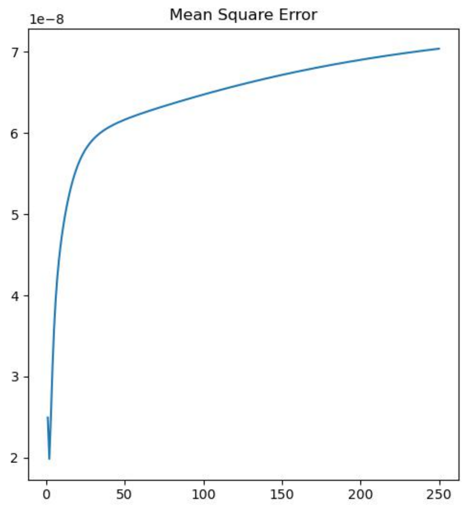
2.4 Gradient descent reconstructed image, ground truth, difference,NUFFT

As the result below shown, the gradient descent reconstructed image has lower resolution but also weaker artifact than NUFFT.



2.5 MSE & L2 norm of the gradient over the iterations.





L2 norm of the gradient deceases and converges at iteration 20.
Mean square error is quite small, although it increases.

3. Iterative image reconstruction with conjugate gradient

3.1. NUFFT

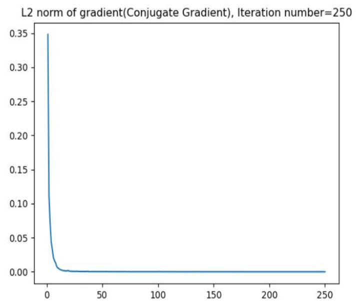
The same as 2.2

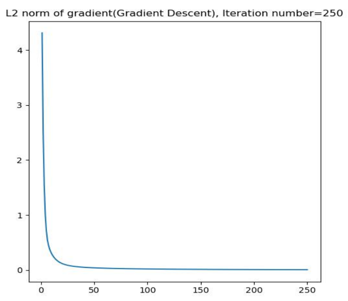
3.2 Conjugate gradient SENSE reconstruction

L2 norm of Conjugate gradient SENSE reconstruction method(CG) and Gradient descent (GD)is as followed:

L2 norm of CG converges at iteration 15 and L2 norm of GD converges at iteration 20. CG method converges a bit earlier.

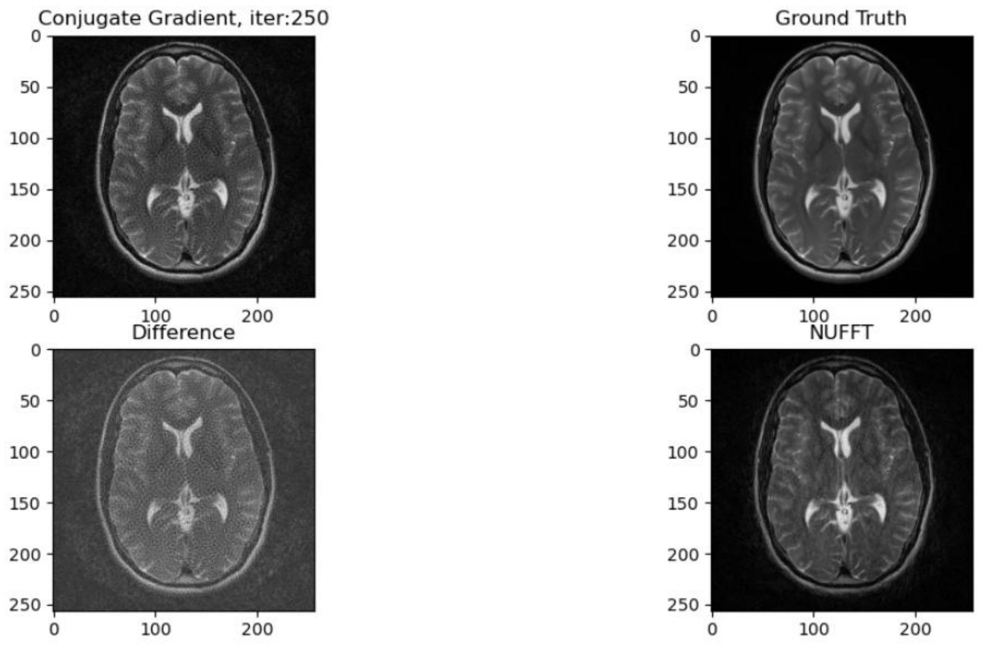
L2 norm of CG is round 10 times smaller than GD, no matter in which iteration(before converge).



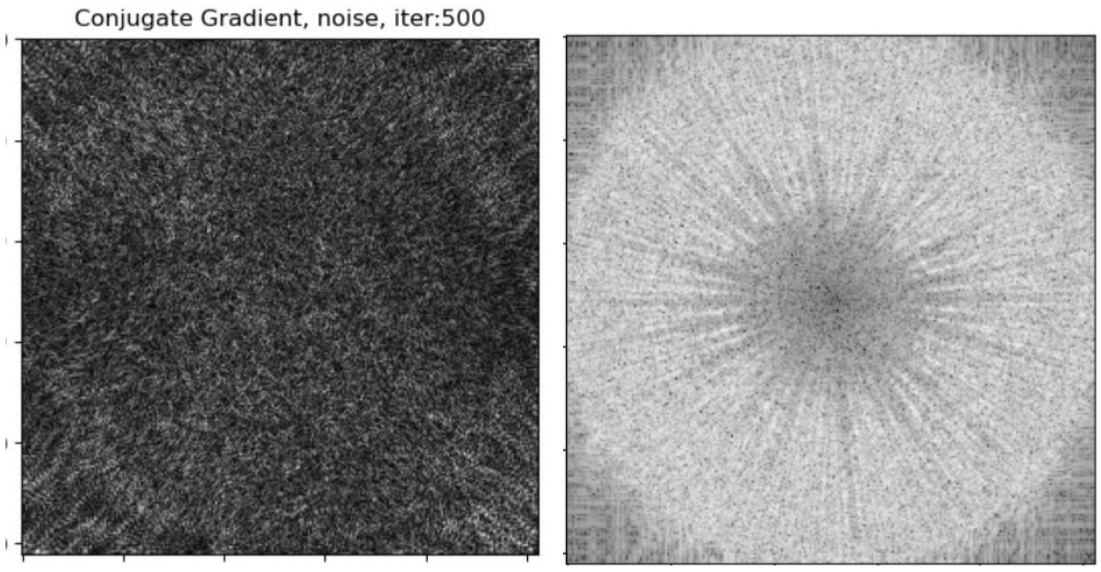


3.3 Conjugate gradient reconstructed image, ground truth, difference,NUFFT

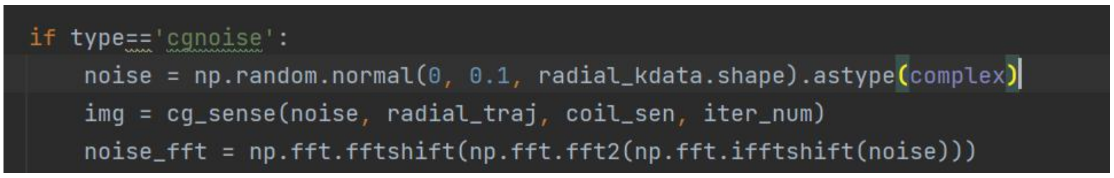
As the result below shown, the conjugate gradient reconstructed image has higher resolution but also stronger artifact than gradient descent method(image above).



3.4 CG reconstruction with noise



The image left is the noise in time domain ; The image right is the noise in kspace.
As the result show, the noise mainly located at the surrounding of radial trajetory instead of in the center.



```
if type=='cgnoise':
    noise = np.random.normal(0, 0.1, radial_kdata.shape).astype(complex)
    img = cg_sense(noise, radial_traj, coil_sen, iter_num)
    noise_fft = np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(noise)))
```