Computational MR imaging

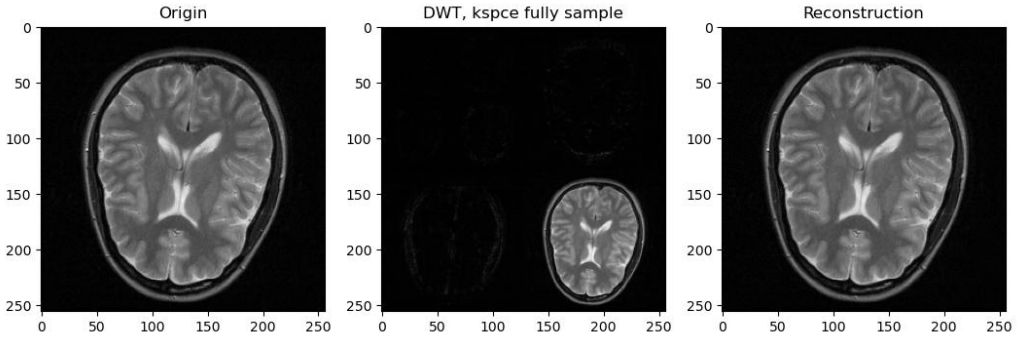
Laboratory 8: Compressed Sensing

Nan Lan

1. Sparsity/compressibility of brain images using the wavelet transform:

i) Wave transfrom

The result below is the wavelet transform, using fully sampled kspace(without compressing).
From the norm1 loss, we can know that there is no great difference between the original image and reconstructed image.




```
norm1_loss = np.linalg.norm((img - recon_img), ord=1)
```

```
Reconstruction error: 9.7399948881677797e-17
```

ii) Compress the brain image by factors 5, 10 and 20

The process of compress sensing algorithm is as followed:

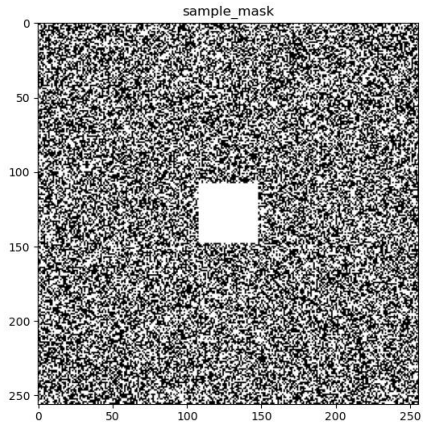
(1) Get the undersampled kspace, according to different compress factor.

```
k_undersample, sample_mask = undersample_kspace(kfull, compress_factor=compress_factor)
```

```python
def undersample_kspace(kfull, compress_factor):
    length = len(kfull)
    left = length//2-20
    right = length//2+20
    sample_mask = np.random.rand(*kfull.shape) > (1 - 1/compress_factor)
    sample_mask[left:right, left:right] = 1
    k_undersample = kfull*sample_mask
    return k_undersample,sample_mask
```

The middle of kspace is fully sampled, the surround of kspace is randomly undersampled

The sample mask is as follow:



(2) Convert the image to wavelet domain

```python
img_wave, s = complex_wd2(ifft2c(k_undersample))
```

(3) Denoise the img_wavelet

Wavelet coefficients represent both space and spatial frequency information. Each band of wavelet coefficients represents a scale (frequency band) of the image. The location of the wavelet coefficient within the band represents its location in space. Threshold the wavelet coefficients retaining only the largest part of the coefficients, according to the compress factor.

```python
img_wave2 = denoise(img_wave, compress_factor=compress_factor, thres_type='soft')
```

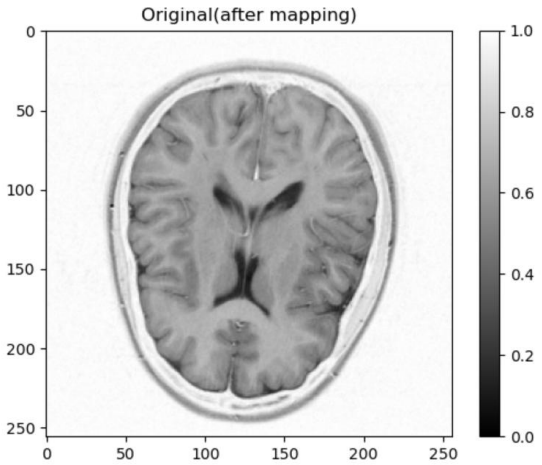
```python
def denoise(img_wave,compress_factor,thres_type):
    m = np.sort(abs(img_wave.ravel()))[::-1]  #descending sort
    ndx = int(len(m) / compress_factor)
    # ndx = int(len(m) * 0.2)

    thr = m[ndx]
    if thres_type=='hard':
        img_wave_thr = HardT(img_wave,thr)   #img_wave * (abs(img_wave) > thr)
    elif thres_type=='soft':
        img_wave_thr = SoftT(img_wave, thr)
    else:
        raise ValueError("thres_type is wrong")
    return img_wave_thr
```

(4) Convert the data from wavelet domain back to image domain

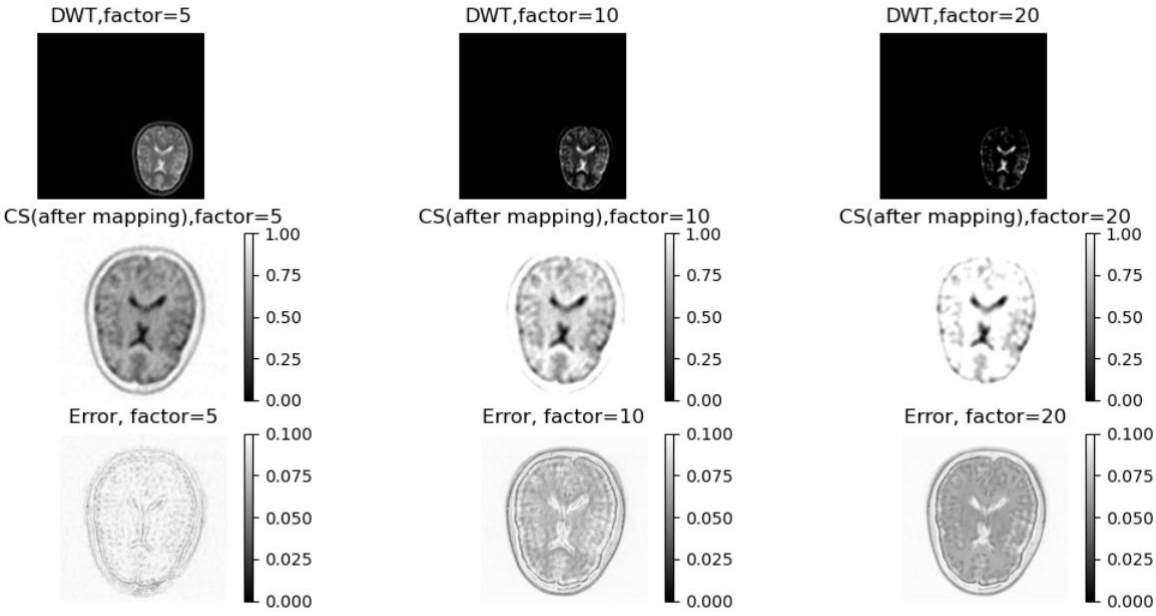
```
recon_img2 = complex_wr2(img_wave2, s)
```

Here is the result.

The result below is the original image(scale: 0-1).



The result below are the Daubechies wavelet transform, reconstructed images(scale: 0-1) and error(scale: 0-0.1).



The following is the evaluation result of different compressing factors. The higher the compress factor, the larger the RMSE.

```
Compress factor is 5
norm1_loss(afer mapping): 16.06190414408648
Root Mean Square Error(afer mapping) is 0.04798

Compress factor is 10
norm1_loss(afer mapping): 51.37823542671337
Root Mean Square Error(afer mapping) is 0.15200

Compress factor is 20
norm1_loss(afer mapping): 74.04465635195074
Root Mean Square Error(afer mapping) is 0.22996
```

2. Compressed sensing reconstruction using iterative soft thresholding:

i) iterative soft-thresholding

The process of iterative soft-thresholding algorithm is as followed:

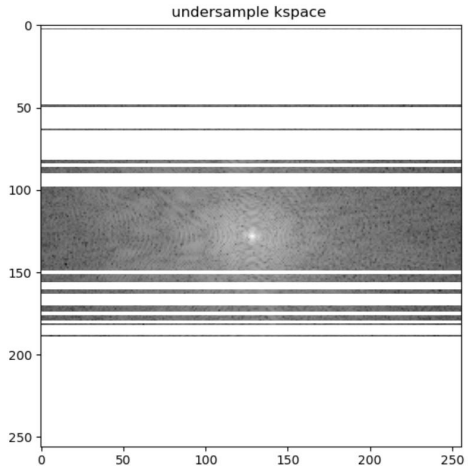
```
def iterative_soft_thresholding(d,lam,compress_factor):
    '''''
    Input:
       d: adquired data, fully samply kspace
       m: original img
    T and Ti are the forward and inverse sparsifying
    '''''
    m = ifft2c(d)  #initial solution
    iter_num = 50
    for i in range(iter_num):
        _, Tm, sample_mask = sparse(kfull, compress_factor)
        cost = np.linalg.norm(fft2c(m) * sample_mask - d, 2) + lam * np.linalg.norm(Tm, 1)
        print("cost in iteration  %1.0f is %1.5f" %(i, cost))
    return m
```

Here is the cost

```
cost in iteration  40 is 67.93353
cost in iteration  41 is 67.93340
cost in iteration  42 is 67.93393
cost in iteration  43 is 67.93356
cost in iteration  44 is 67.93318
cost in iteration  45 is 67.93324
cost in iteration  46 is 67.93350
cost in iteration  47 is 67.93362
cost in iteration  48 is 67.93356
cost in iteration  49 is 67.93366
```

ii) kacc

The following is the kacc




```
num = np.count_nonzero(np.abs(kacc[:,1])>1e-8)
print('sample line number ', num)
acc_factor = len(kacc)/(len(kacc) - num)
print('acc_factor ',acc_factor)
```

```
sample line number  114
acc_factor  1.8028169014084507
```

The acceleration factor is 1.8