ELEN 6886

Homework 1

xw2501

Xiaoning Wan

For details about the codes, see 'readme' file or notes in code scripts.

Exercise 1

a.

as given in the question, represent x and U as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad U = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_n^T \end{bmatrix}$$

Then, for each entry of x we have

$$E[x_i] = 0$$

Consequently, for each entry of Ux

$$E\left[u_i^T x_i\right] = E\left[u_i^T\right] E[x_i] = 0$$

Thus, **the mean of Ux is 0**.

As for the covariance matrix,

$$\Sigma_x = E(xx^T) - \mu\mu^T = E(xx^T) = I$$

And for the covariance matrix of Ux,

$$\Sigma_{Ux} = E((Ux)(Ux)^T) - \mu\mu^T = E((Ux)(Ux)^T)$$
$$= E(Uxx^T U^T)$$
$$= UE(xx^T)U^T$$
$$= UU^T = I$$

The covariance matrix of Ux is I

Thus, Ux and x are identically distributed.

b.

As proven in question 'a' above, Ux and x have identical distribution. Thus, to prove $\frac{x}{\|x\|_2}$ and $\frac{Ux}{\|Ux\|_2}$ have the same distribution, only need to prove $\|x\|_2 = \|Ux\|_2$.

$$\|x\|_2 = x^T x = x^T I x = x^T U^T U x = (Ux)^T(Ux) = \|Ux\|_2$$

Thus, distribution of $\frac{x}{\|x\|_2}$ is invariant to rotation.

c.

The i_{th} column of matrix $\mathrm{orth}(N)$ is given as $m_i = \frac{n_i - \sum_{j=1}^{i-1} \frac{<n_i, m_j>}{<m_j, m_j>} m_j}{\left\| n_i - \sum_{j=1}^{i-1} \frac{<n_i, m_j>}{<m_j, m_j>} m_j \right\|}$.

Define $\mathrm{orth}(N) = [m_1 \quad m_2 \quad \cdots \quad m_p]$ and $Q\mathrm{orth}(N) = [Qm_1 \quad Qm_2 \quad \cdots \quad Qm_p]$.

Suppose $\mathrm{orth}(QN) = [m'_1 \quad m'_2 \quad \dots \quad m'_p]$.

First, to prove $m'_1 = Qm_1$

As given in the formula

$$m'_1 = \frac{Qn_1 - \sum_{j=1}^{0-1} \frac{< Qn_1, m'_j >}{< m'_j, m'_j >} m'_j}{\left\| Qn_1 - \sum_{j=1}^{0-1} \frac{< Qn_1, m'_j >}{< m'_j, m'_j >} m'_j \right\|} = \frac{Qn_1}{\|n_1\|} = Qm_1$$

And for any other columns m'_i $(i \neq 1)$, suppose that for all $j < i$, we already have $m'_j = Qm_j$, then for m'_i

$$m'_i = \frac{Qn_i - \sum_{j=1}^{i-1} \frac{< Qn_i, m'_j >}{< m'_j, m'_j >} m'_j}{\left\| Qn_i - \sum_{j=1}^{i-1} \frac{< Qn_i, m'_j >}{< m'_j, m'_j >} m'_j \right\|} = \frac{Qn_i - \sum_{j=1}^{i-1} \frac{< Qn_i, Qm_j >}{< Qm_j, Qm_j >} Qm_j}{\left\| Qn_i - \sum_{j=1}^{i-1} \frac{< Qn_i, Qm_j >}{< Qm_j, Qm_j >} Qm_j \right\|}$$

$$= \frac{Qn_i - \sum_{j=1}^{i-1} \frac{< n_i, m_j >}{< m_j, m_j >} Qm_j}{\left\| Qn_i - \sum_{j=1}^{i-1} \frac{< n_i, m_j >}{< m_j, m_j >} Qm_j \right\|} = \frac{Q(n_i - \sum_{j=1}^{i-1} \frac{< n_i, m_j >}{< m_j, m_j >} m_j)}{\left\| Q(n_i - \sum_{j=1}^{i-1} \frac{< n_i, m_j >}{< m_j, m_j >} m_j) \right\|} = m_i$$

Now that we've proven

i. if for all $j < i$, there is $m'_j = Qm_j$ then $m'_i = Qm_i$.
ii. $m'_1 = Qm_1$

So for all $0 \le i \le p$, we can say $m'_i = Qm_i$, hence $\mathrm{orth}(QN) = Q\mathrm{orth}(N)$.

d.

```
$Sample\ matrix\ N \in \mathbb{R}^{n\times k}\ i.i.d\ in\ a\ random\ distribution$
for $i = 1,2 \dots n$
    $m_i \leftarrow n_i - \sum_{j=1}^{i-1} \frac{<n_i, m_j>}{<m_j, m_j>} m_j$
end
for $i = 1,2 \dots n$
    $m_i \leftarrow \frac{m_i}{\|m_i\|}$
```

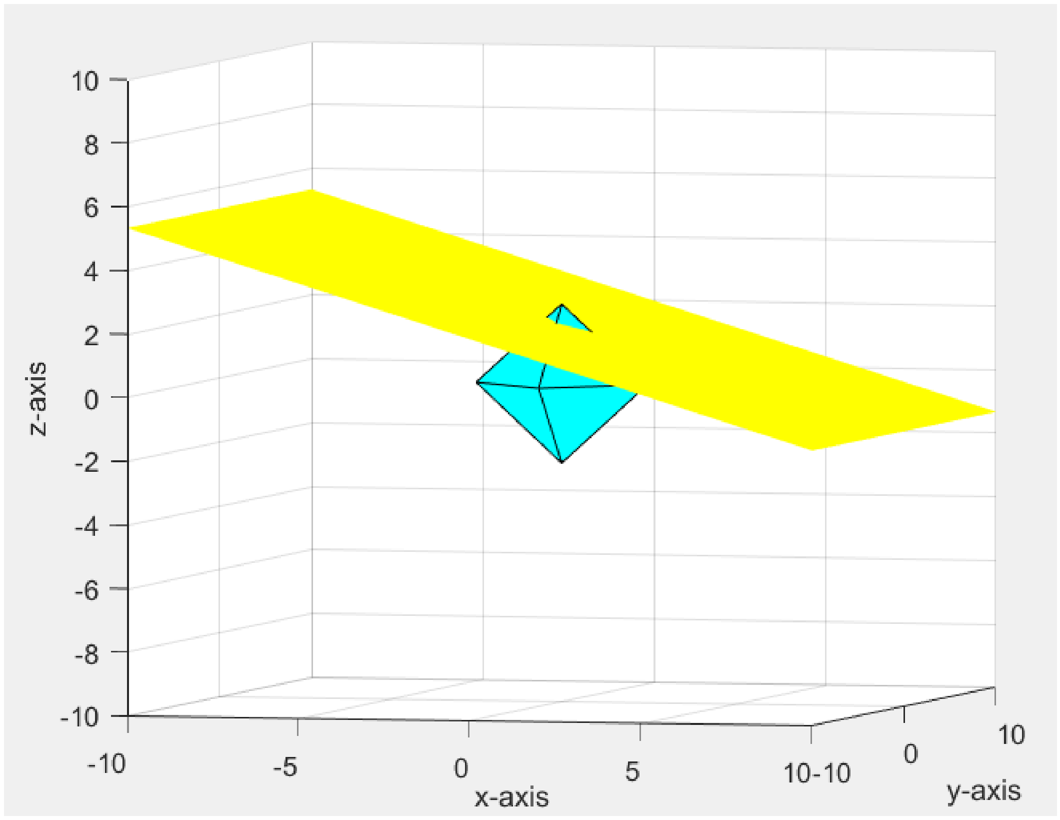
```
end
return $[m_1, m_2, \ldots m_k]$
```

Exercise 2

a.

code is uploaded.

A screenshot is like this



Blue one is the cross polytype. Yellow region is the affine set.

b.

i.

code is uploaded.

I implemented three algorithms to find the projection on l1-ball. Reference paper (http://www.optimization-online.org/DB_FILE/2014/08/4498.pdf and http://stanford.edu/~jduchi/projects/jd_ss_ys_paper.pdf). The first one runs in $O(nlog(n))$ time complexity, the second one runs in $O(n)$ time complexity, but I am not sure about the third one.

All these three algorithms work good. You may try any one by uncomment that part and comment the rest.

ii.

The algorithm I designed is basically like this.

```
Initialize $l, w$ with random value
while not converge
    $w \leftarrow POCS(w, l, A, y)$
    $l \leftarrow l + \eta[norm(w) - l]$
end
```

$Ax = y$ is the affine set.

$POCS(w, l, A, y)$ is the algorithm used in b.i.

l is the l1 norm and η is the updating rate.

One thing to note is that, **this algorithm requires the initial l to be very small, that is to have the cross-polytope and affineset not intersect.**

Thus, by calling the algorithm in b.i. we have a point lying on the affine set and has a relatively small l1 norm. And then we update l, our prediction value. When the l1 norm of the point and our prediction value matches, we can say that now the cross-polytope and affine set intersects with each other. And l is the smallest l1 norm for a point in the affine set.

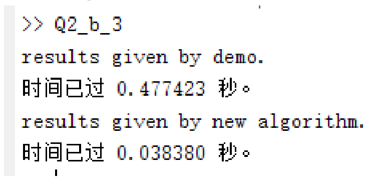
iii.

The test is based on following parameters.

```
m = 100;
n = 60;
non_empty_entry = 24;
```
with all none zero parameters randomly generated.

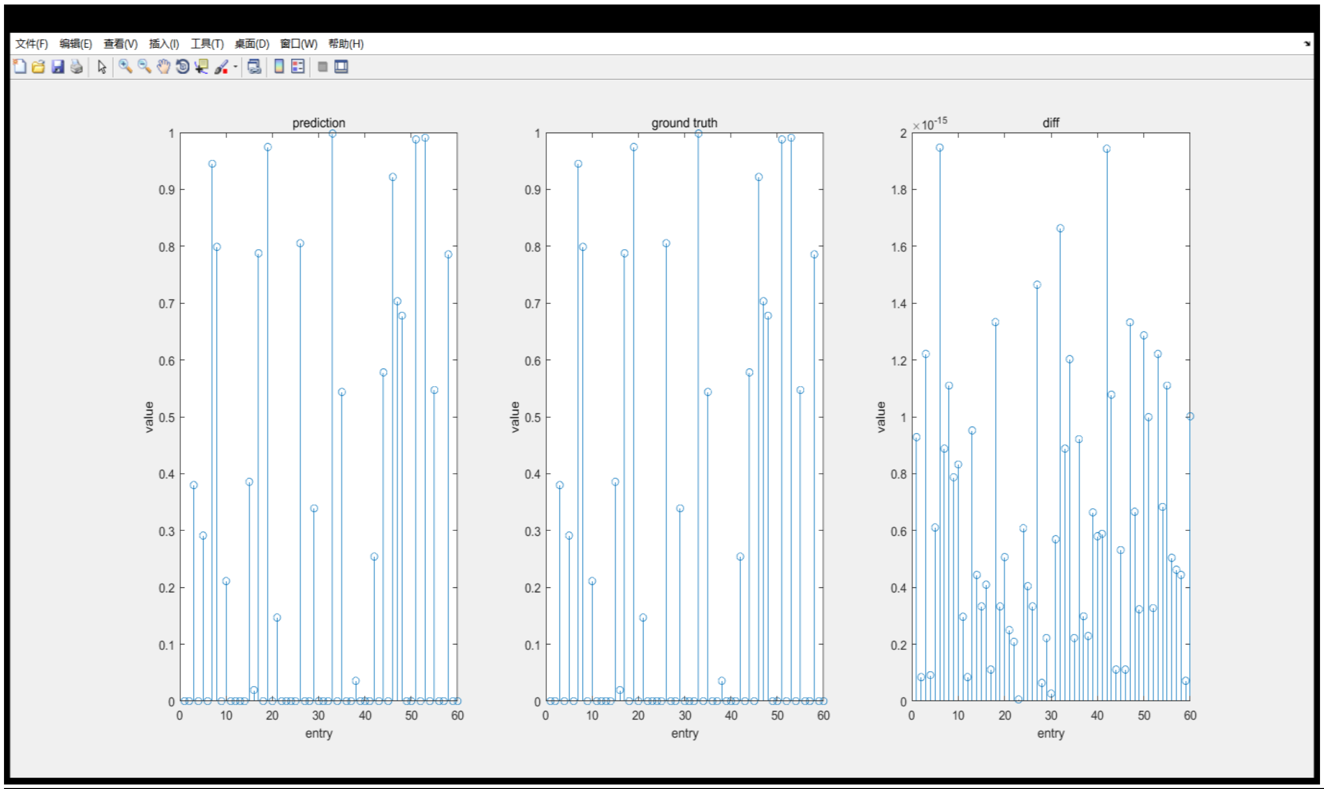
Result is like this

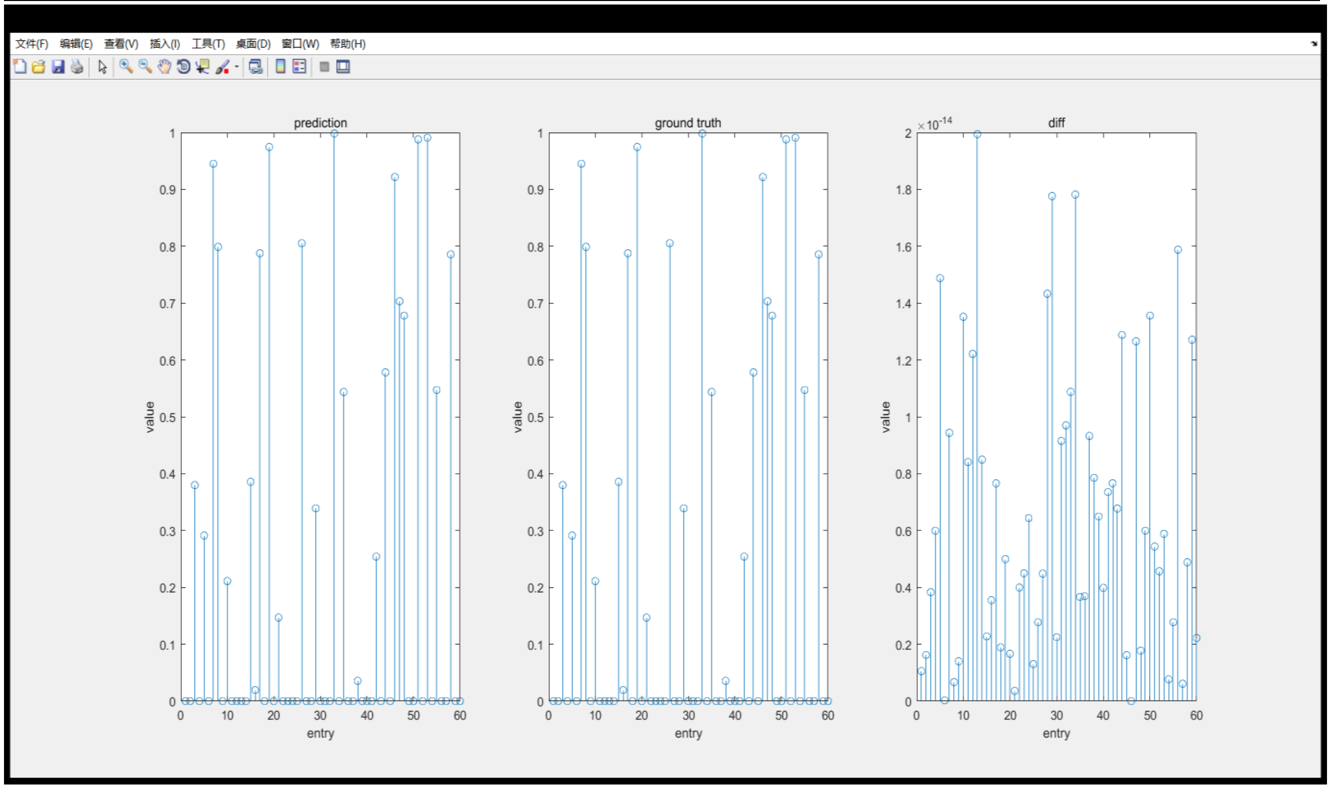
- running time



```
>> Q2_b_3
results given by demo.
时间已过 0.477423 秒。
results given by new algorithm.
时间已过 0.038380 秒。
```

  First is the time used by demo, second is the time used by this algorithm. The new algorithm is 10 times faster than the demo code. (the unit of time is second)
- running result





First graph is the result given by demo, second graph is given by new algorithm. Both of them have a very good accuracy.

c.

The updating rate is turned down for better illustration. So the running time might be longer.

Exercise 3

a.

The 2D-DFT is basically like this

$$F[m,n] = \frac{1}{UV}\sum_{u=0}^{U-1}\sum_{v=0}^{V-1} X[u,v]\exp(-j2\pi(\frac{m}{U}u + \frac{n}{V}v))$$

In this case, $U = V = Z$ so the right form should be

$$F[m,n] = \frac{1}{Z^2}\sum_{u=0}^{Z-1}\sum_{v=0}^{Z-1} X[u,v]\exp(-\frac{j2\pi}{Z}(mu + nv))$$

Proof of linearity

$$\mathrm{DFT}(\alpha X + \beta Y) = \frac{1}{UV}\sum_{u=0}^{U-1}\sum_{v=0}^{V-1} (\alpha X[u,v] + \beta Y[u,v])\exp(-j2\pi(\frac{m}{U}u + \frac{n}{V}v))$$

$$= \alpha\frac{1}{UV}\sum_{u=0}^{U-1}\sum_{v=0}^{V-1} X[u,v]\exp(-j2\pi(\frac{m}{U}u + \frac{n}{V}v)) + \beta\frac{1}{UV}\sum_{u=0}^{U-1}\sum_{v=0}^{V-1} Y[u,v]\exp(-j2\pi(\frac{m}{U}u + \frac{n}{V}v))$$

$$= \alpha F_X[m,n] + \beta F_Y[m,n]$$

Thus, this DFT operation is linear.

And the elementwise multiplication is obviously linear. Thus, $S_U \circ F$ is linear.

To show the projection matrix, we need to reshape matrix $X \in \mathbb{C}^{Z\times Z}$ to a vector $X_R \in \mathbb{C}^{Z^2}$.

Where $X = \begin{bmatrix} x_{11} & \cdots & x_{1Z} \\ \vdots & \ddots & \vdots \\ x_{Z1} & \cdots & x_{ZZ} \end{bmatrix}$ and $X_R = \begin{bmatrix} x_{11} \\ \vdots \\ x_{1Z} \\ x_{21} \\ \vdots \\ x_{2Z} \\ \vdots \\ \vdots \\ x_{ZZ} \end{bmatrix}$.

Thus, the operator can be expressed as $A \in \mathbb{C}^{Z^2\times Z^2}$

$$Y = AX = U_\Omega F X$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_U \end{bmatrix} = \begin{bmatrix} u_{\Omega_1}^T \\ u_{\Omega_2}^T \\ \vdots \\ \vdots \\ u_{\Omega_U}^T \end{bmatrix} \begin{bmatrix} a_{11}^{11} & \dots & a_{1Z}^{11} & a_{21}^{11} & \dots & a_{2Z}^{11} & \dots & \dots & a_{ZZ}^{11} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ a_{11}^{1Z} & \dots & a_{1Z}^{1Z} & a_{21}^{1Z} & \dots & a_{2Z}^{1Z} & \dots & \dots & a_{ZZ}^{1Z} \\ a_{11}^{21} & \dots & a_{1Z}^{21} & a_{21}^{21} & \dots & a_{2Z}^{21} & \dots & \dots & a_{ZZ}^{21} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ a_{11}^{2Z} & \dots & a_{1Z}^{2Z} & a_{21}^{2Z} & \dots & a_{2Z}^{2Z} & \dots & \dots & a_{ZZ}^{2Z} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\ a_{11}^{ZZ} & \dots & a_{1Z}^{ZZ} & a_{21}^{ZZ} & \dots & a_{2Z}^{ZZ} & \dots & \dots & a_{ZZ}^{ZZ} \end{bmatrix} \begin{bmatrix} x_{11} \\ \vdots \\ x_{1Z} \\ x_{21} \\ \vdots \\ x_{2Z} \\ \vdots \\ \vdots \\ x_{ZZ} \end{bmatrix}$$

Where $a_{uv}^{mn} = \frac{1}{Z^2} exp(-\frac{j2\pi}{Z}(mu + nv))$ and u_{Ω_i} is a vector with one entry 1 and the rest 0, sort of like one hot encoding, the 1 entry indicates the relevant data is selected.

And consequently, the adjoint map given in matrix form is

$$A^T = (U_\Omega F)^T = F^T U_\Omega^T$$

Where U_Ω^T is the transpose of U_Ω, which is a mapping from a vector in $\mathbb{C}^m$ to $\mathbb{C}^{Z^2}$. To be specific, it put the member back in to a matrix at the position where it is sampled from. And the rest members of matrix are all 0.

And F^T is the transpose of F, it is the inverse fourier transformation with a scale. And the scale is UV.

Now we can write this adjoint in function form.

$$A^* = UVF^{-1} \circ U_\Omega^*$$

And the last thing is to find $(AA^*)^{-1}$. To find this mapping operator, it would be much easier if we write it back in to matrix form.

$$(AA^T)^{-1} = \left(UVU_\Omega FF^{-1}U_\Omega^T\right)^{-1} = (UVI_{mm})^{-1} = \frac{1}{UV}I$$

This means the transform $(AA^T)^{-1}$ scales the input m dimension vector by $\frac{1}{UV}$ times.

If we interpret in a function way, mapping $AA^* = U_\Omega \circ F \circ (UVF^{-1}) \circ U_\Omega^* = UV(U_\Omega \circ F \circ F^{-1} \circ U_\Omega^*)$. And as mentioned above, F and F^{-1}, U_Ω and U_Ω^* will cancel each other out, thus, the mapping basically scales the input by UV. Consequently, its inverse should be scaling the input by $\frac{1}{UV}$.

Finally, we can insert what we computed into the projection formula

$$z = x - A^*(AA^*)^{-1}(Ax - y)$$

$$z = x - (UVF^{-1} \circ U_\Omega^*) \cdot \frac{1}{UV} \cdot (U_\Omega Fx - y)$$

Thus, we can use the projection gradient descend algorithm to find the solution.

```
while $not\ converge$
        $X \leftarrow \underset{X}{\operatorname{argmin}}(\|X\|_1)$
        $X \leftarrow X - (UVF^{-1} \circ U_\Omega^*) \cdot \frac{1}{UV} \cdot [U_\Omega F(X) - Y]$
end
```

One thing to note here.

The algorithm I implemented not only included fourier transform and sampling, but also has wavelet decomposition. However, I am not very familiar with wavelet decomposition, so I will not solve for the adjoint of wavelet decomposition.

I used function ‘wavelet_coeffs’ and ‘resconstruct_image’ in MRI demo code to realize wavelet decomposition and its adjoint mapping, wavelet recovery.

Though I could not prove they are mapping and adjoint pair theoretically, we can test it by experiment. I specially wrote a script for it. Details will be explained in ‘readme’.

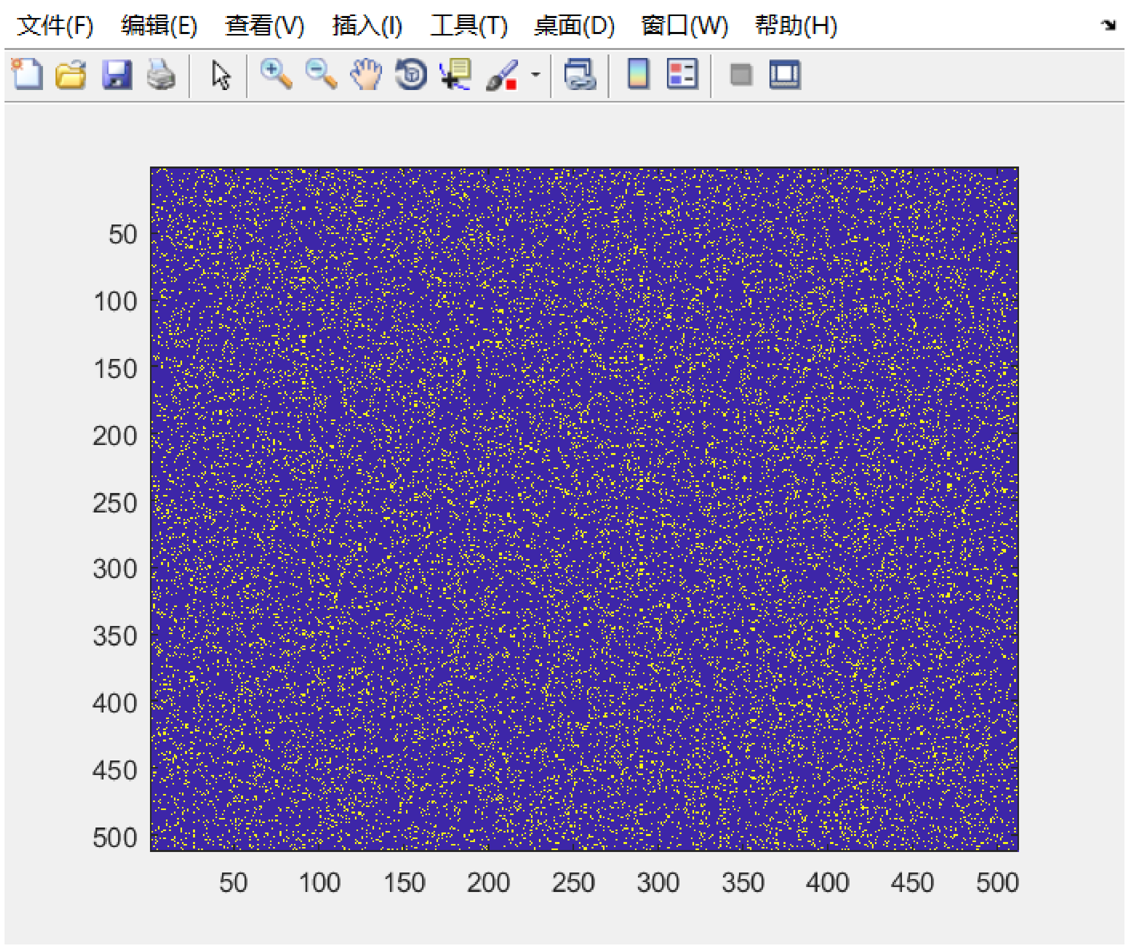
b.

Just as mentioned above, the algorithm I used is gradient descend with projection. Again, I used the function 'minimize_L1_proj_subgrad' in demo. I made some changes, details will be included in 'readme'.

1.

For this question, just randomly choose desired number of entries is enough.

Here is an illustration of how the data are sampled



Yellow points indicate the data of that pixel is sampled. We can see that the data are sampled uniformly from the figure.

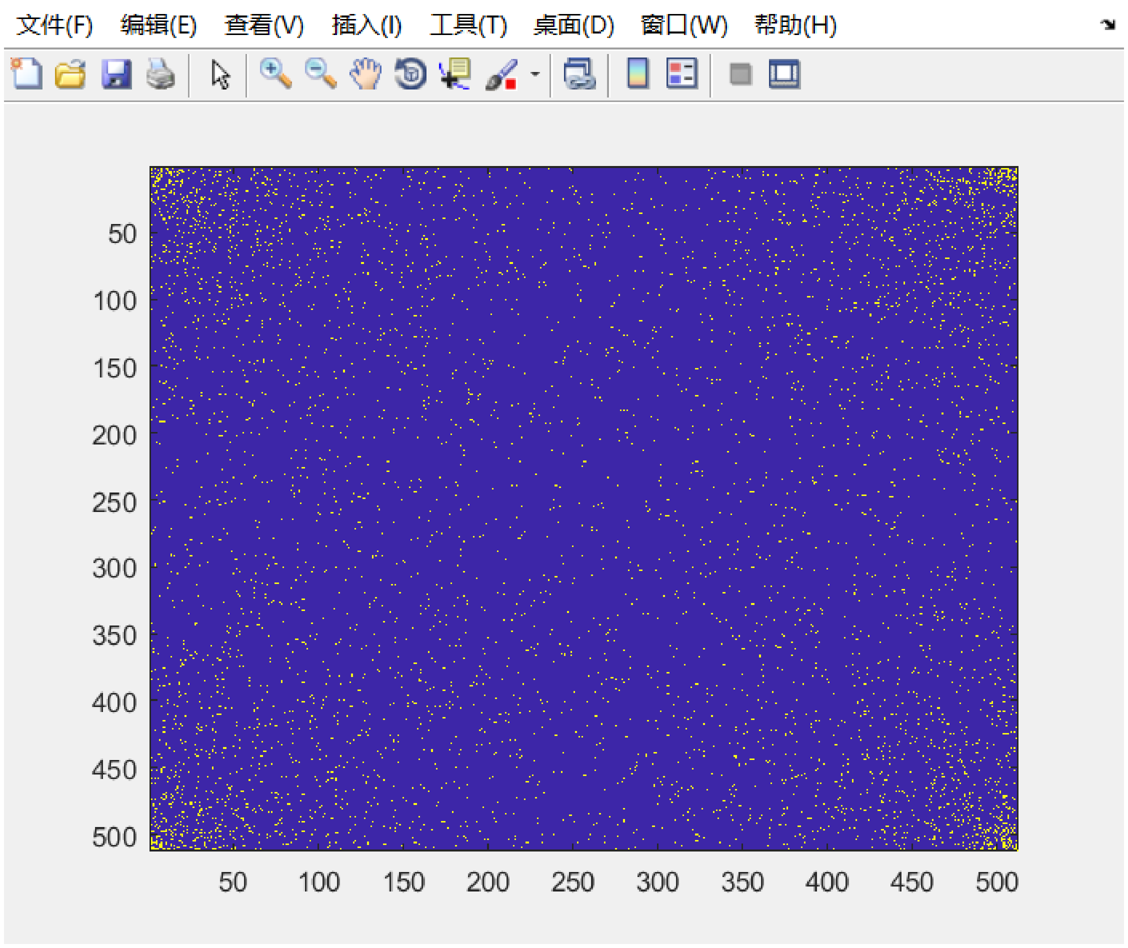
2.

For variable density, we want to sample more low frequency data and less high frequency data, as most of the information is concentrated at low frequency.

Thus, we need to sample with some weight, and the weight I used is the reciprocal of the distance to central.

$$weight = \frac{1}{dist}$$

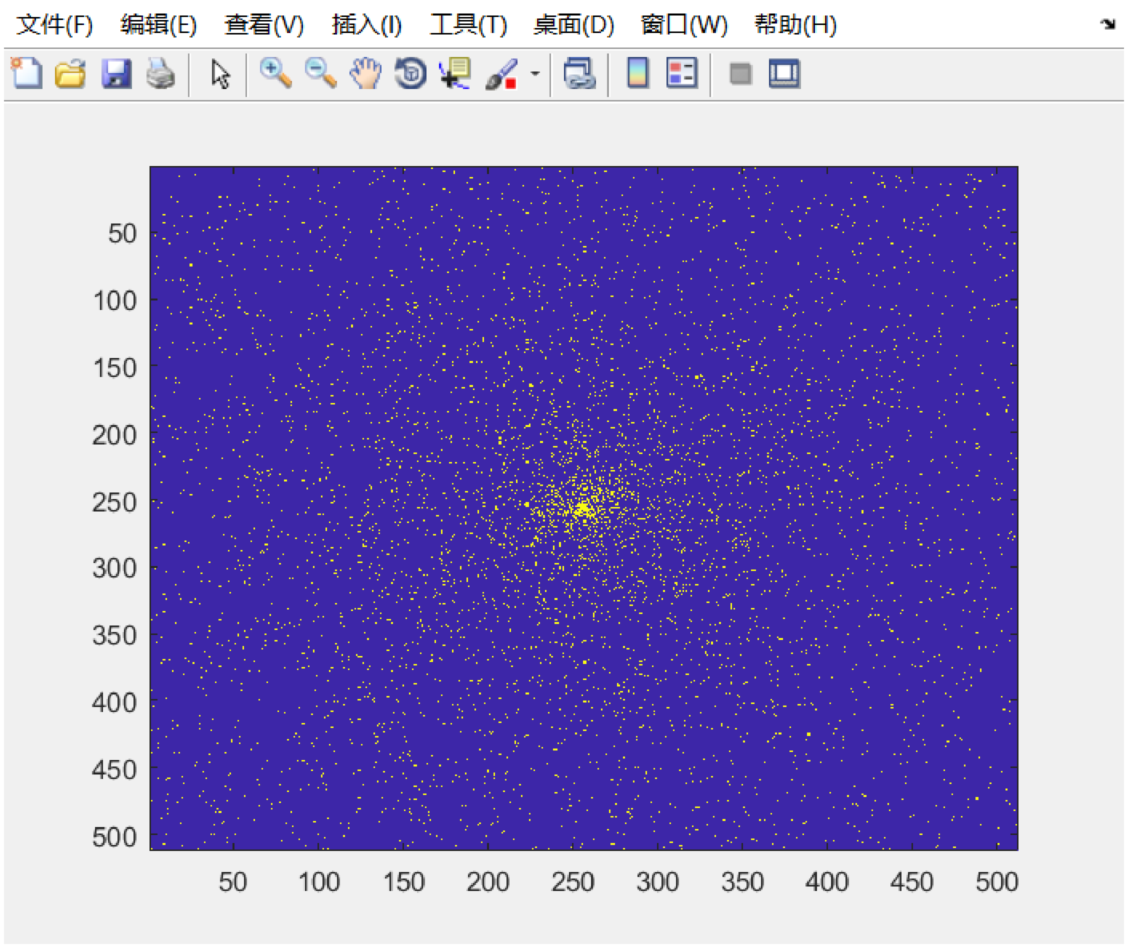
Here is an illustration of how the data are sampled



You can see that more data are sampled at the corner and much less at the center.

Note that the density should be higher at low frequency, and in matlab, when you call ‘fft2’ function, the low frequency part is assigned to the corners instead of the center unless you call another function named ‘fftshift’. In my codes, I did not call this ‘fftshift’ function, and thus the high density part is not at center but at corners.

For your information, the shifted sampling graph is like this



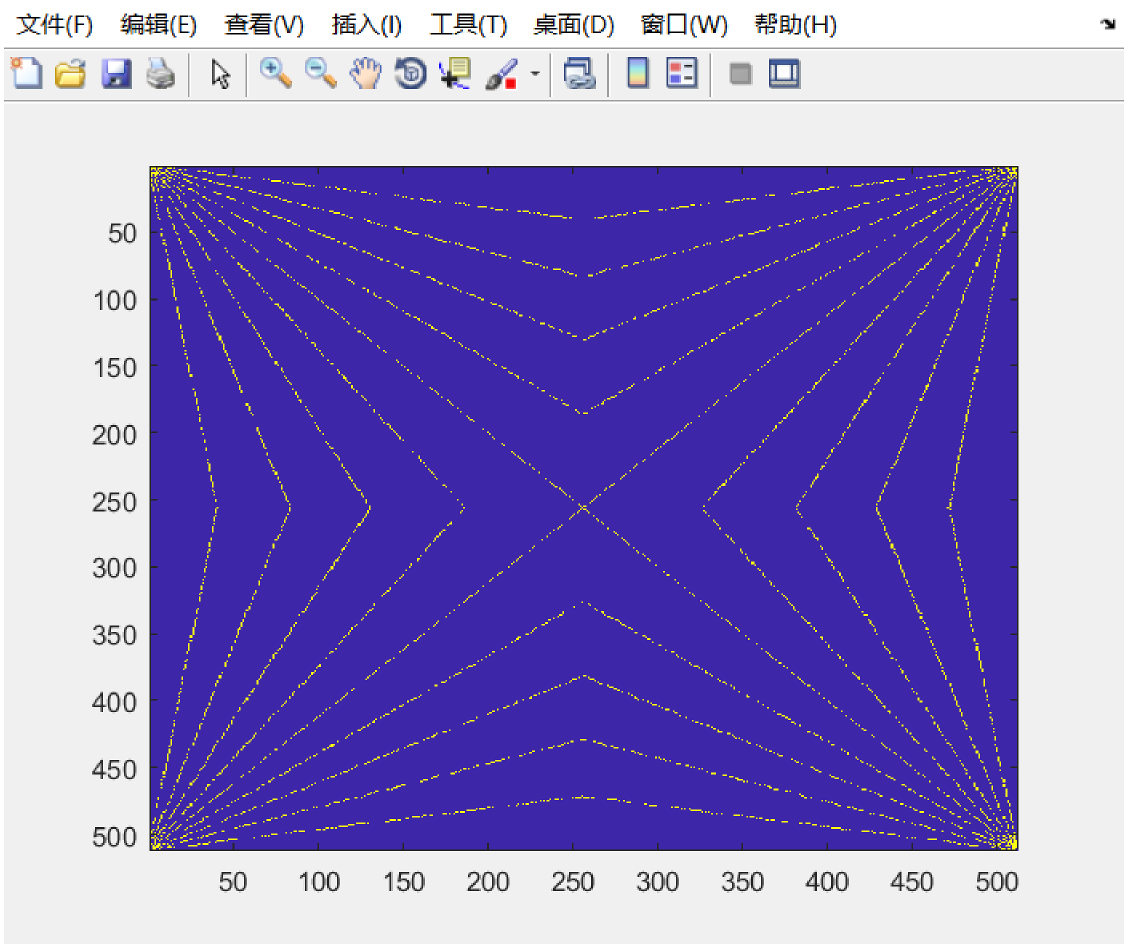
Now it is much clearer that more data are sampled at low frequency.

3.

For this method of sampling, the method is pretty straight forward, just set an angle growth step, and then sweep the angle from 0 to 2π.

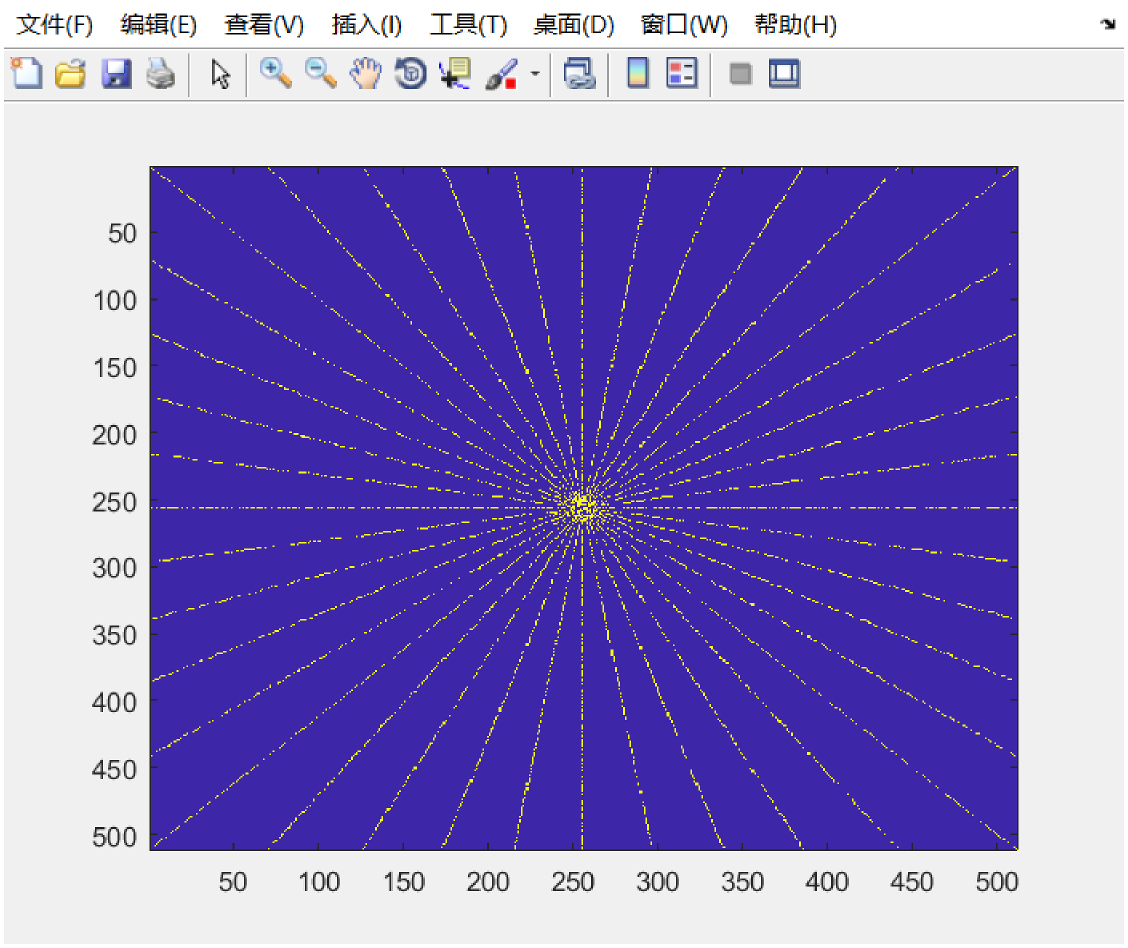
To realize only sampling specific number of data we want, I first label all the point on radial lines to 1, and then randomly sample data within these points.

Here is an illustration of how the data are sampled



Again, as I did not call the ‘fftshift’ function, the center of radial lines is not at the center but at the corners.

For your information, the shifted sampling graph is like this



You may notice that some points on the lines are not sampled, this is because in order to compare between these three means of sampling, we need to control the sampled points to be the same. And thus, I set the lines first and then sample desired number of points lying on these lines.

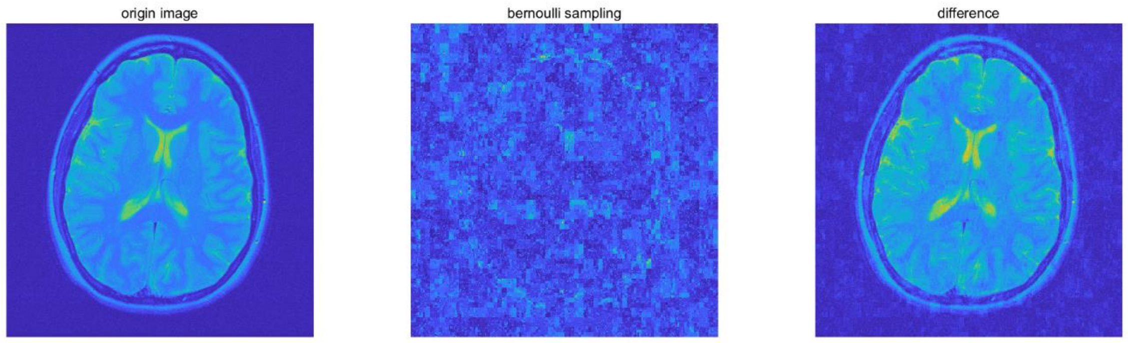
C.

Let’s analyze their performance individually first.

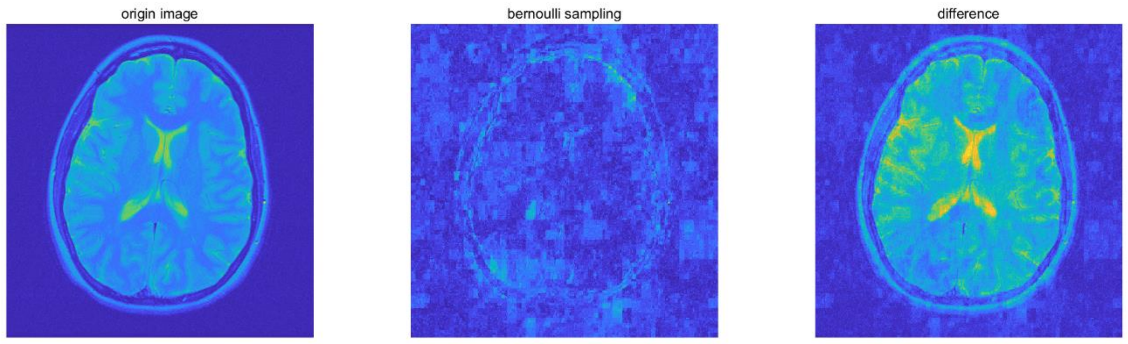
I used theta to denote for the sampling rate (how much proportion of data is sampled).

Bernoulli

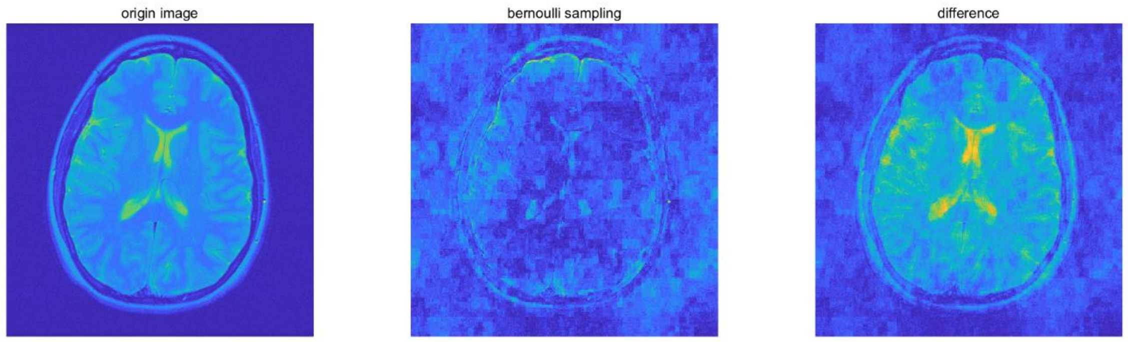
theta = 0.1



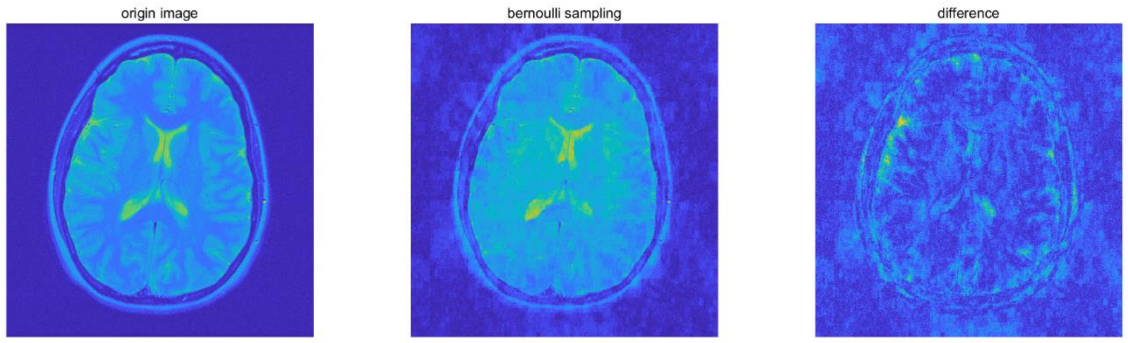
theta = 0.2



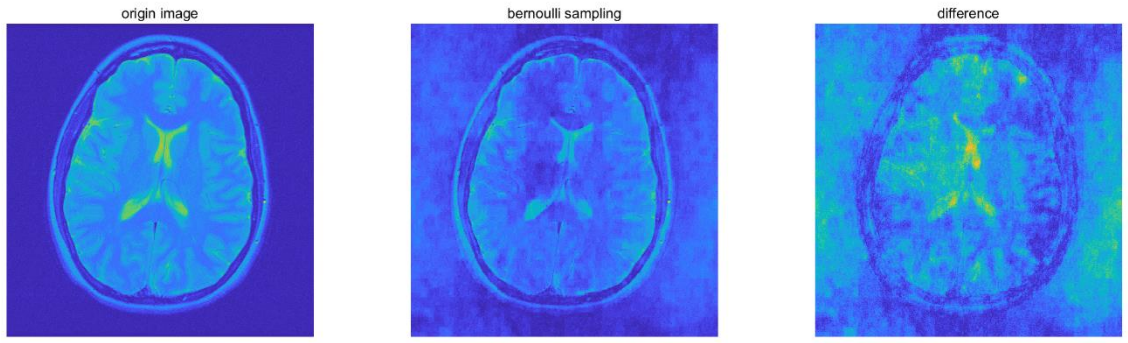
theta = 0.3



theta = 0.4



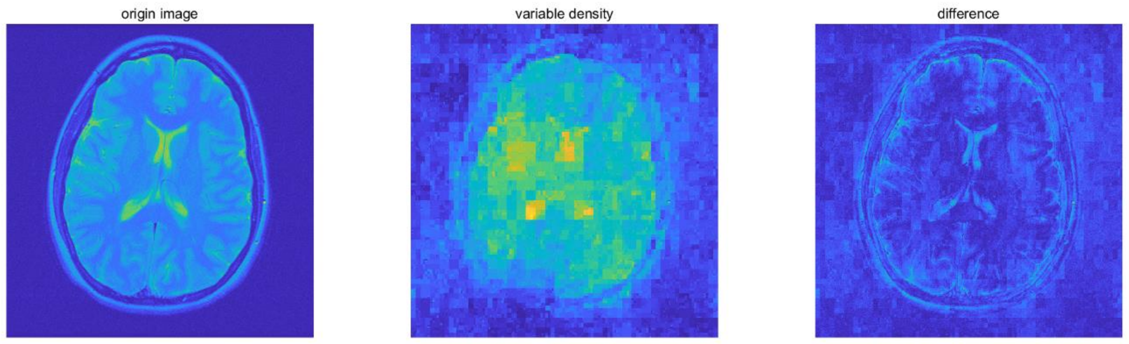
theta = 0.5



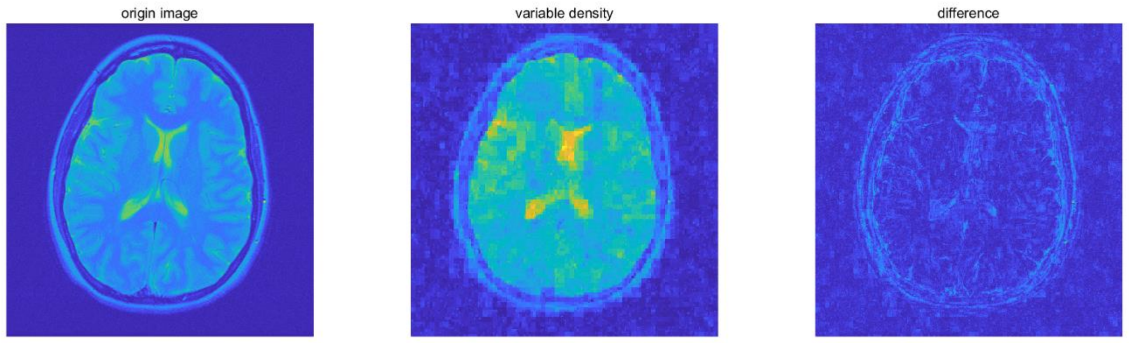
As can be seen, the recovered image is quite good after sampling rate is equal or higher than 0.4. As this method samples data uniformly, when sampling rate is low, we lose a lot of information in low frequency, which is of crucial importance in image recovering.

Variable Density

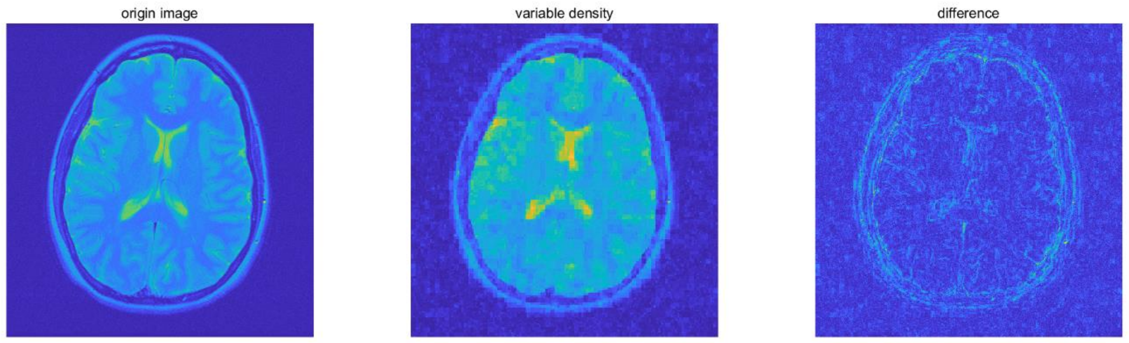
theta = 0.025



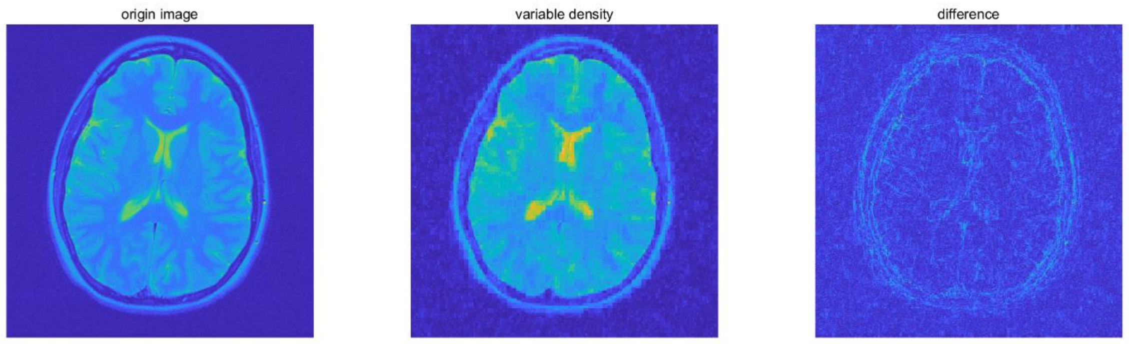
theta = 0.05



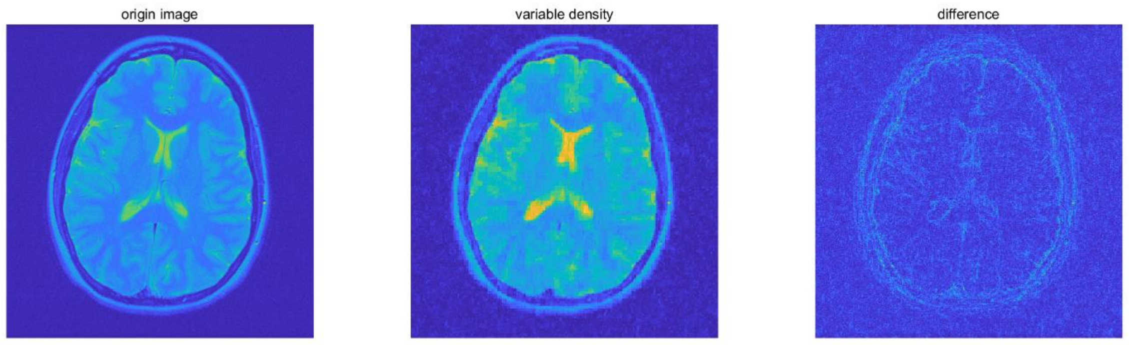
theta = 0.075



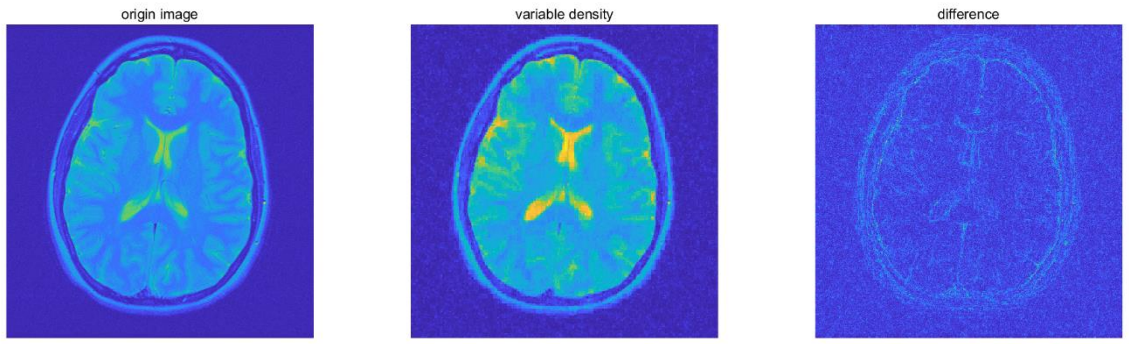
theta = 0.1



theta = 0.125



theta = 0.15

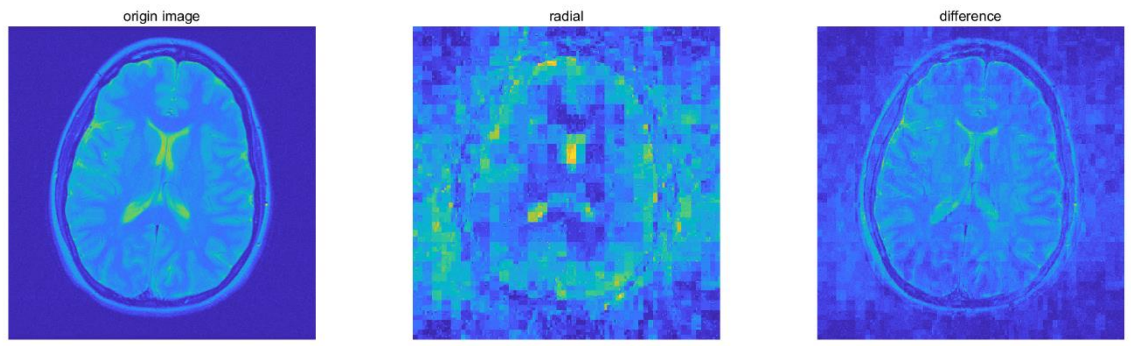


In this case, the recovered image is much better than the former one, when sample rate is 0.75, we can roughly see the shape of the origin image. This is because this method mainly samples points at low frequency, and most of the information of origin image is at low frequency. Thus, this method requires much less proportion of data to reconstruct image.

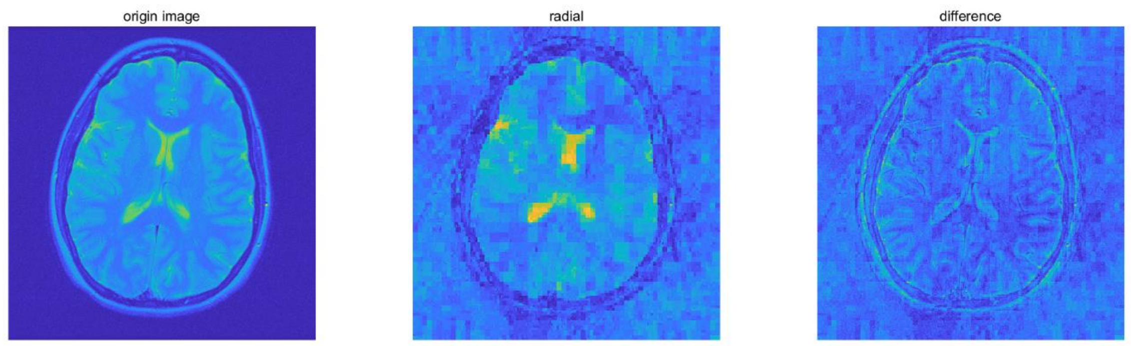
However, as this method samples very less at high frequency, some edges in the recovered image is not very smooth, this can obviously be seen in the 'difference' image.

Radial

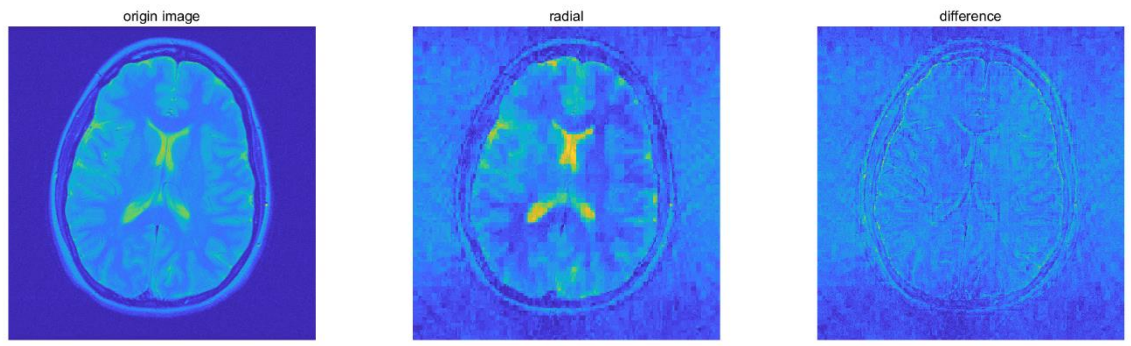
theta = 0.025



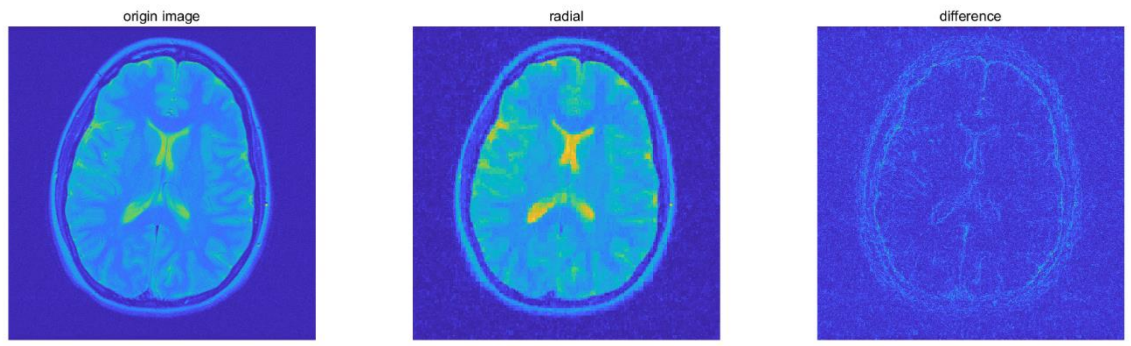
theta = 0.05



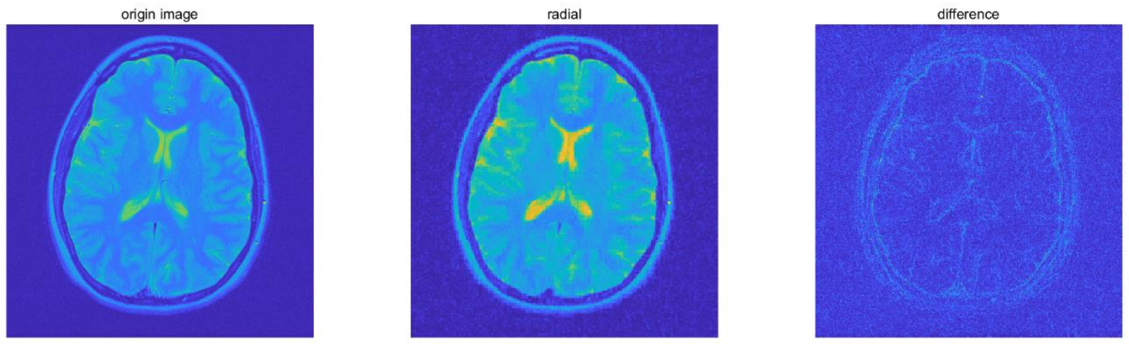
theta = 0.075



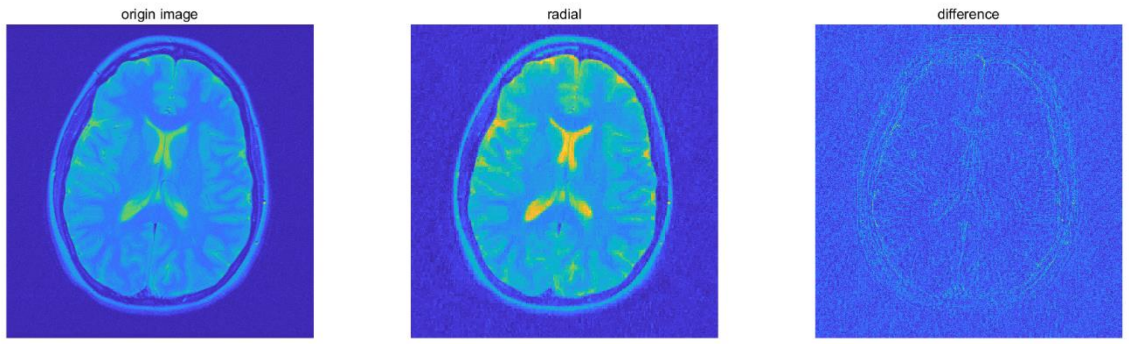
theta = 0.1



theta = 0.125



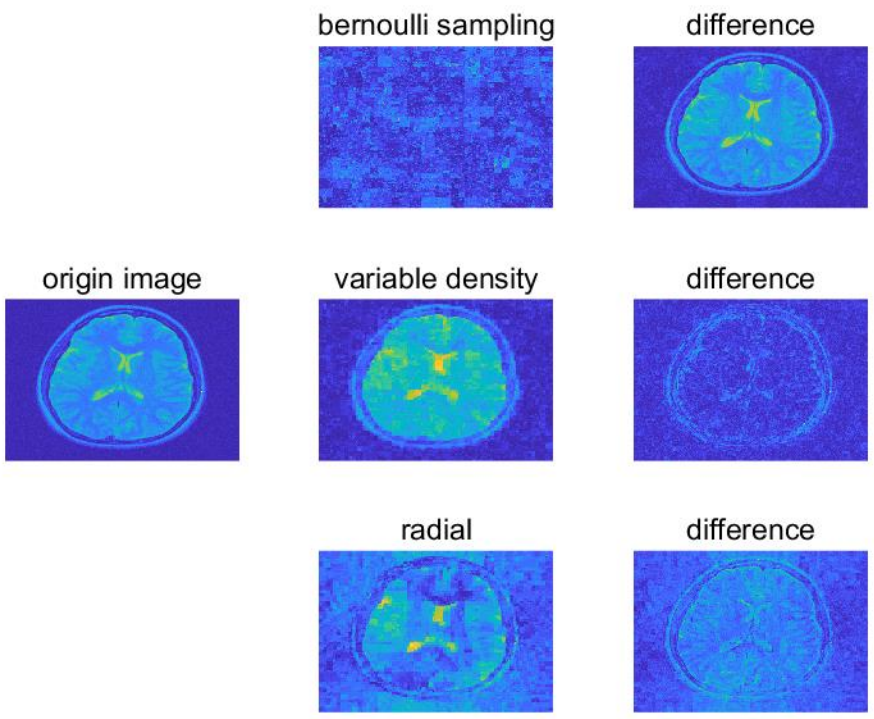
theta = 0.15



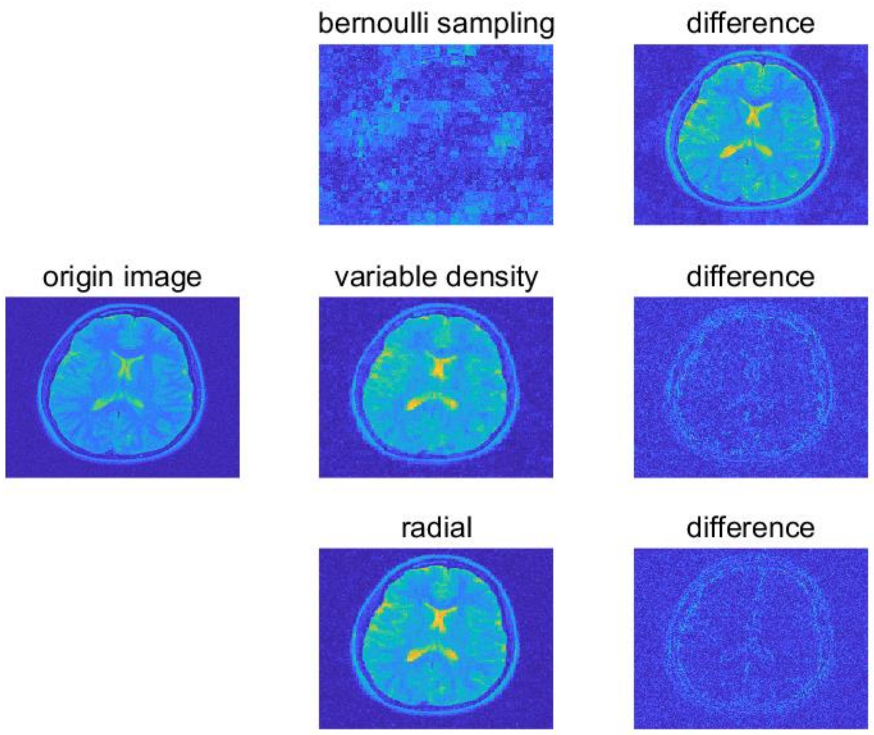
Not only does this sampling method give good performance with relatively small proportion of samples, but also gives a better performance in high frequency compare to the variable density sampling. The edge of recovered images are smoother.

Compare

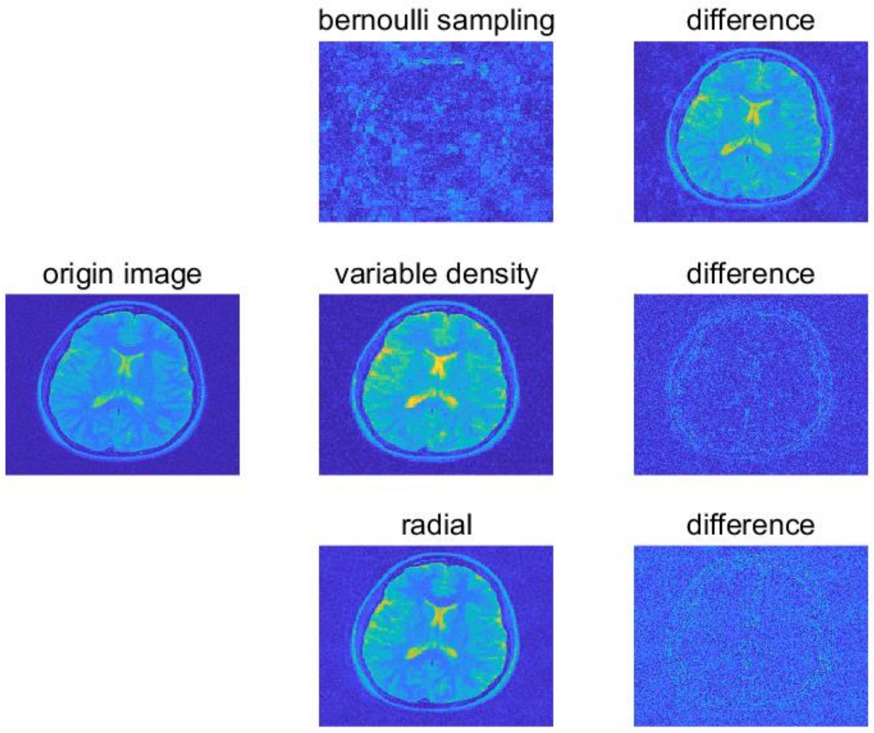
theta = 0.5



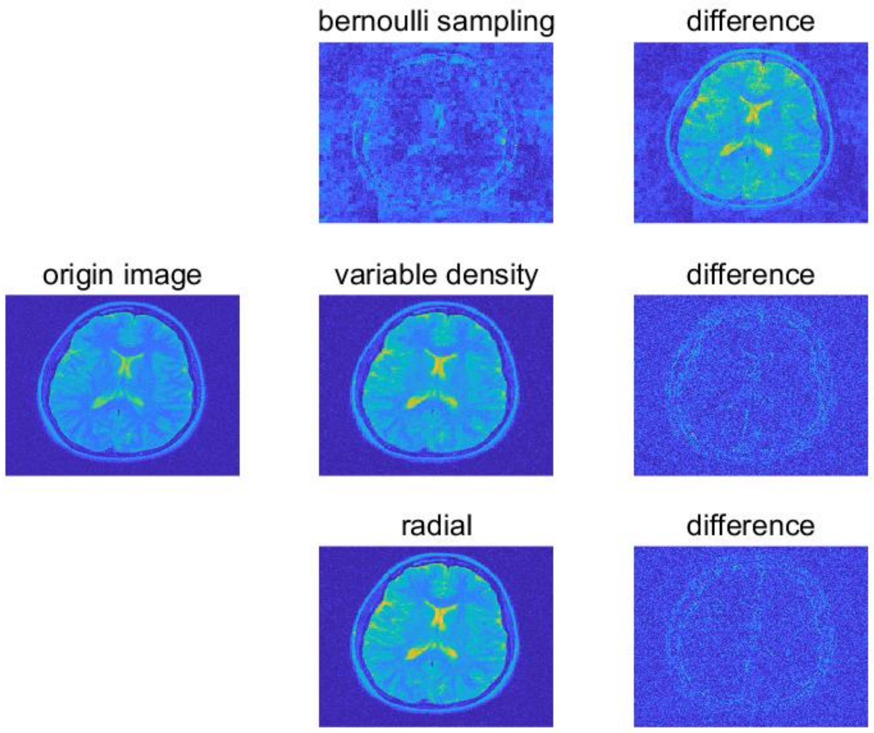
theta = 0.1



theta = 0.15



theta = 0.2



According to the result, the variable density has the best performance when sampling rate is really low (at 0.05), and when sampling rate is low but not extremmely low (0.1 and 0.15), radial sampling has the best performance. When the sampling rate further grows, variable density sampling and radial sampling have similar performance. As for the bernoulli sampling, its performance is bad among all these four tests. Only when the sampling rate is very high (at 0.4), bernoulli sampling reconstruction will give an acceptable image.

The result of reconstruction of algorithms I implemented is not as good as keeping the top-magnitude wavelet. When 0.07 of top-magnitude wavelet is kept, the reconstructed image is nearly the same as origin image. While in bernoulli sampling, the reconstructed image has a really poor performance and in veriable density sampling, radial sampling, the reconstructed images basically show the structure of the brain, but there is still an obvious loss at high frequency part (the egdes are not smooth).

Given that Z is sparse in wavelet space, and as wavelet level increases, the wavelet space of Z becomes even sparser. Thus, keeping the top-magnitude entries is basically keep all the information useful in recovering the origin image. That is why MRI demo has a good performance.

I am not sure what the hardware of an MRI machine would look like, but I guess it is the radial sampling. Since the number of samples is the number of measurments we need to take, and when taken same number of samples, radial sampling will give the best performance. So, we can say that radial sampling would need least number of measurments to recover the image. However, if the difficulty of taking measurment at different entry of the matrix is different, it is another story.