# Computational MR imaging

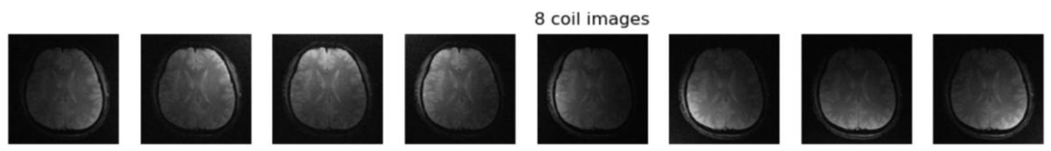
## Laboratory 5: Image space parallel imaging

**Nan Lan**

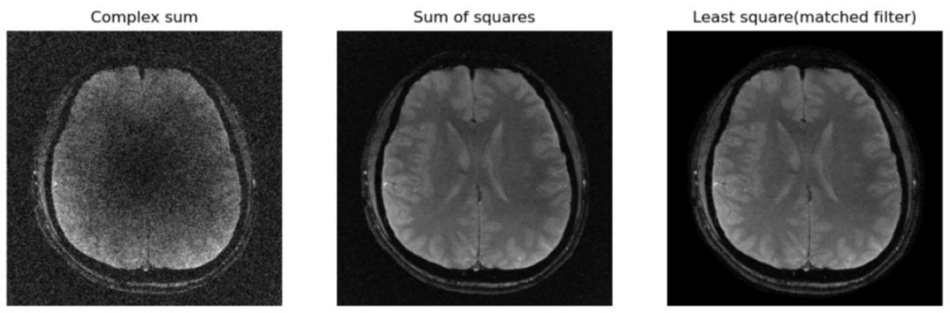
## 1. Multicoil combination

### (1) Multicoil combination algotrithms

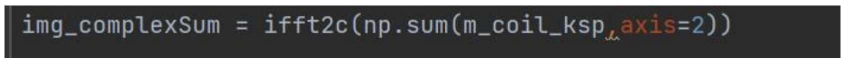
The results below shows MRI coil images from 8 coils.



The results below shows outputs of different Multicoil combination algotrithms.
Due to the phase cancellation in ''Complex sum'', the result is quite noisy.
The images reconstructed from ''Sum of square'' and ''matched filter'' are clear and similar.

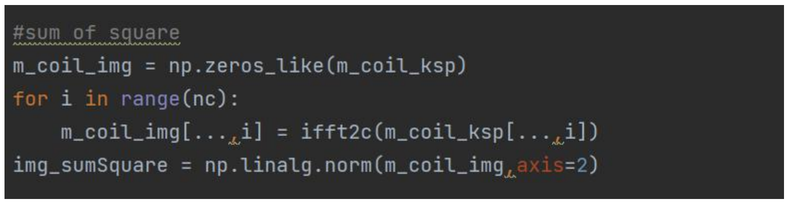


**The process of complex sum is as follow:**
Sum up all the pixel along the coil direction.



```
img_complexSum = ifft2c(np.sum(m_coil_ksp,axis=2))
```

**The process of sum of square is as follow:**
Use L2-norm along coil direction to combine the coil images.



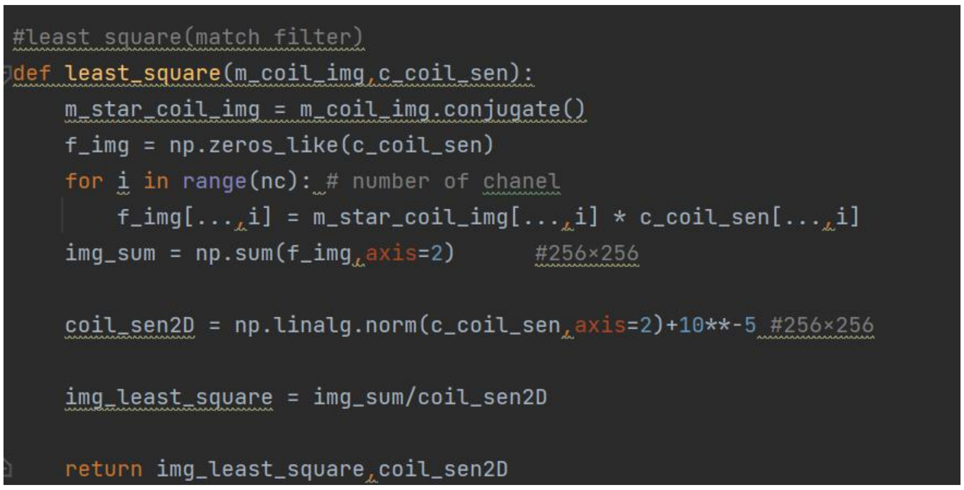
```
#sum of square
m_coil_img = np.zeros_like(m_coil_ksp)
for i in range(nc):
    m_coil_img[...,i] = ifft2c(m_coil_ksp[...,i])
img_sumSquare = np.linalg.norm(m_coil_img,axis=2)
```

**The process of match filter is as follow:**

$$f(r)=\frac{\sum_{i=1}^{N_c} c_i^*(r)m_i(r)}{\sqrt{\sum_{l=1}^{N_c}|cl(r)|^2}}$$

$m_i(r)$: single coil images
$c_i(r)$: coil sensitivities

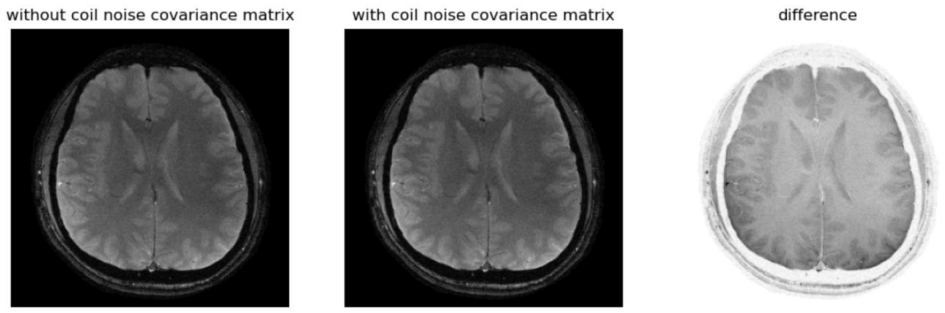
Use the equation above to combine the coil image, also along the coil direction.



```
#least square(match filter)
def least_square(m_coil_img,c_coil_sen):
    m_star_coil_img = m_coil_img.conjugate()
    f_img = np.zeros_like(c_coil_sen)
    for i in range(nc): # number of chanel
        f_img[...,i] = m_star_coil_img[...,i] * c_coil_sen[...,i]
    img_sum = np.sum(f_img,axis=2)      #256×256

    coil_sen2D = np.linalg.norm(c_coil_sen,axis=2)+10**-5 #256×256

    img_least_square = img_sum/coil_sen2D

    return img_least_square,coil_sen2D
```

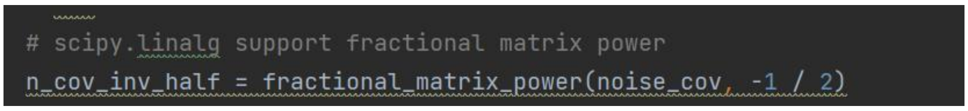
## (2) Coil noise covariance matrix

The results below shows the effect of coil noise covariance matrix.
The correction of coil noise covariance matrix doesn't have much effect on the recontructed image.


without coil noise covariance matrix
with coil noise covariance matrix
difference

**The process of the Coil noise covariance matrix is as follow:**

1) Calculate the half inverse of the coil noise covariance matrix $\Psi^{-½}$

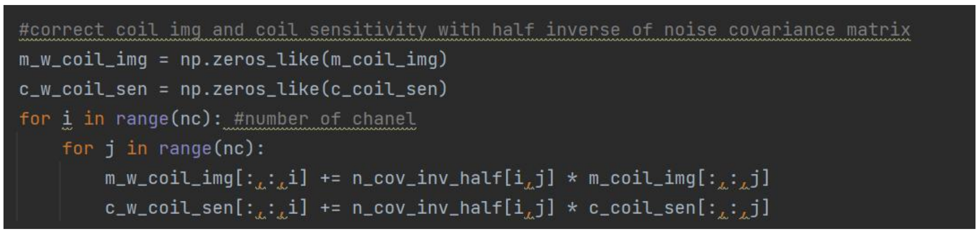


```
# scipy.linalg support fractional matrix power
n_cov_inv_half = fractional_matrix_power(noise_cov, -1 / 2)
```

2) Pre-whitening: correct coil img and coil sensitivity with half inverse of noise covariance matrix, based on the equation below.

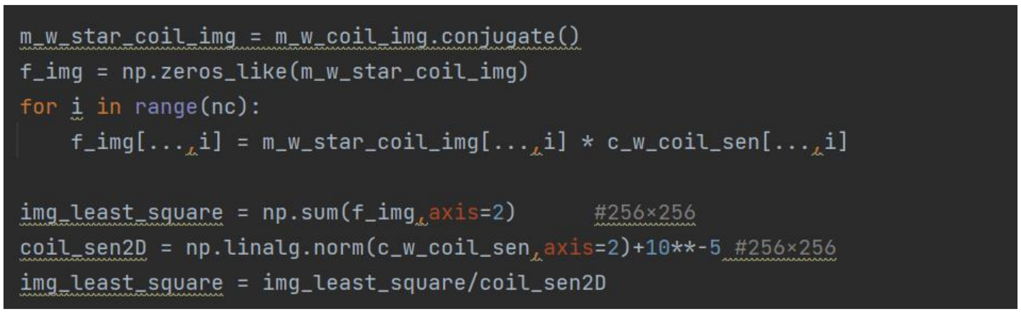
$$\mathbf{m}_w = \mathbf{\Psi}^{-\frac{1}{2}}\mathbf{m}$$

$$\mathbf{C}_w = \mathbf{\Psi}^{-\frac{1}{2}}\mathbf{C}$$



```
#correct coil img and coil sensitivity with half inverse of noise covariance matrix
m_w_coil_img = np.zeros_like(m_coil_img)
c_w_coil_sen = np.zeros_like(c_coil_sen)
for i in range(nc): #number of chanel
    for j in range(nc):
        m_w_coil_img[:,:,i] += n_cov_inv_half[i,j] * m_coil_img[:,:,j]
        c_w_coil_sen[:,:,i] += n_cov_inv_half[i,j] * c_coil_sen[:,:,j]
```

3) The same step as matched filter algorithm.

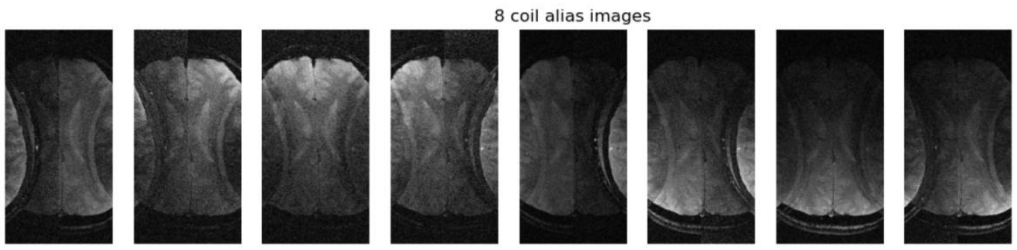


```
m_w_star_coil_img = m_w_coil_img.conjugate()
f_img = np.zeros_like(m_w_star_coil_img)
for i in range(nc):
    f_img[...,i] = m_w_star_coil_img[...,i] * c_w_coil_sen[...,i]

img_least_square = np.sum(f_img,axis=2)      #256×256
coil_sen2D = np.linalg.norm(c_w_coil_sen,axis=2)+10**-5 #256×256
img_least_square = img_least_square/coil_sen2D
```

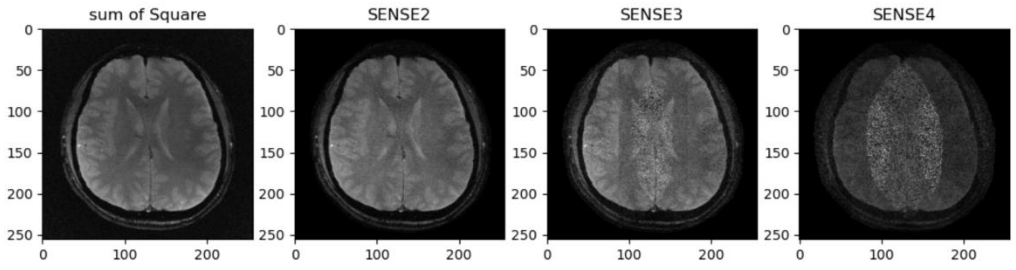
## 2. Cartesian SENSE reconstruction and g-factor

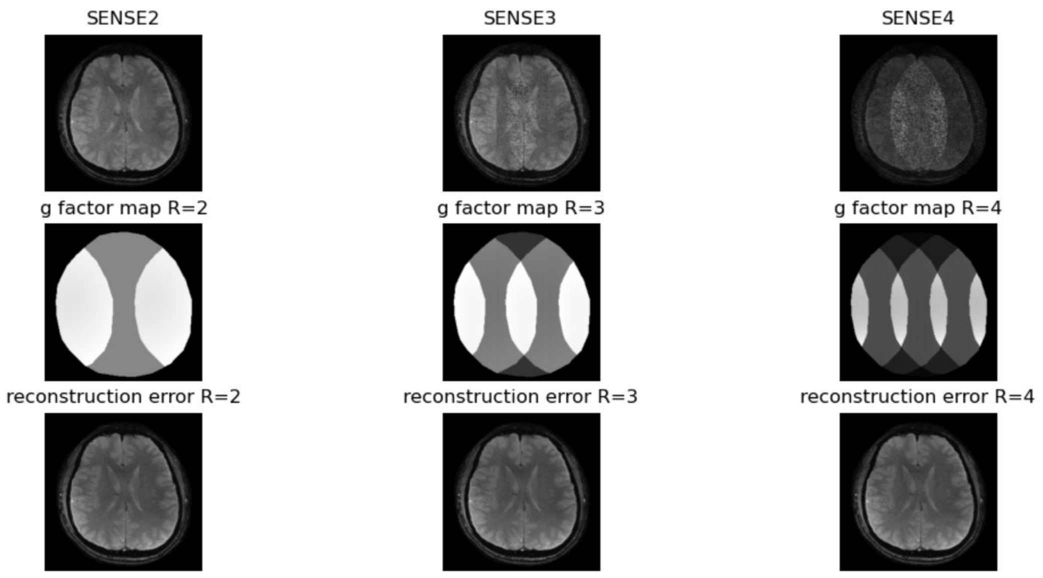
### (1) SENSE algorithm

The results below shows MRI coil alias images from 8 coils, with g-factor of 2.



The results below shows outputs of SENSE algorithm, with different accelaration factor(R=2,3,4).





The larger the accelaration factor R, the more noise the MRI image. Besides, the noise in SENSE recontructed image is not uniformly distributed throughout the image. Instead, the noise is mainly located in the middle region, where the pixel overlap there.

Parallel imaging (PI) seem "noisy" . This is a direct result of the fact that the main purpose of PI technology is to reduce imaging time. In SENSE algorithm so, fewer data points are collected and averaged, so the signal-to-noise ratio (SNR) is reduced accordingly. Therefore, the SNR of the PI sequence is always lower than the equivalent non-PI sequence.

The SNR equation is as follow.

$$SNR_{parallel} = \frac{SNR}{g\sqrt{R}}$$

## (2) G-factor and SNR loss and RMSE

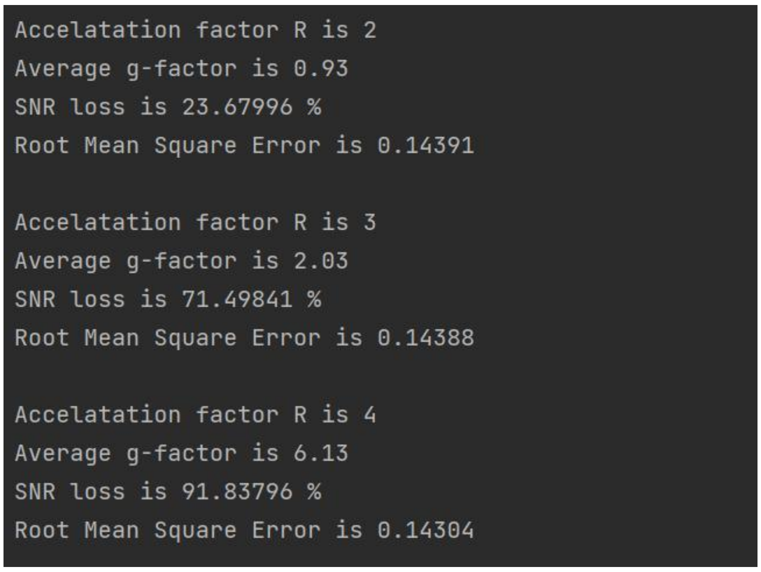
**Average g-factor:** mean g-factor exclude the pixels outside the brain

**SNR loss**: SNR_loss = (1-1/(g_mean*np.sqrt(r)))*100

**RMSE**:

$$\mathrm{RMSE}(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

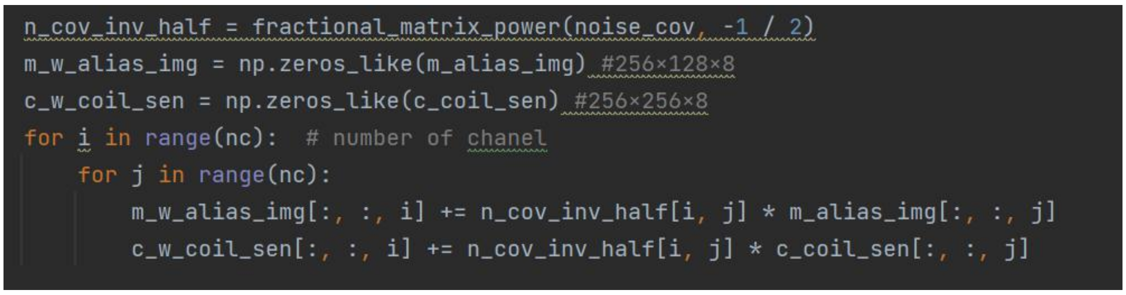
The result of average g-factor, SNR loss and RMSE are as following:



```
Accelatation factor R is 2
Average g-factor is 0.93
SNR loss is 23.67996 %
Root Mean Square Error is 0.14391

Accelatation factor R is 3
Average g-factor is 2.03
SNR loss is 71.49841 %
Root Mean Square Error is 0.14388

Accelatation factor R is 4
Average g-factor is 6.13
SNR loss is 91.83796 %
Root Mean Square Error is 0.14304
```

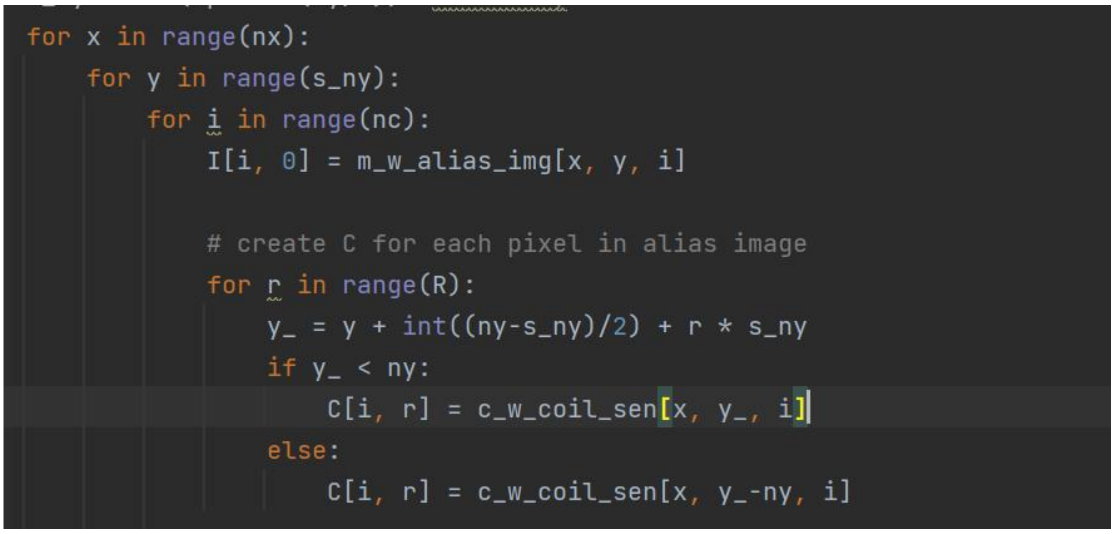
**The principle of the SENSE is as follow:**

1) correct coil img and coil sensitivity with the half inverse of the coil noise covariance matrix $\Psi^{-1/2}$



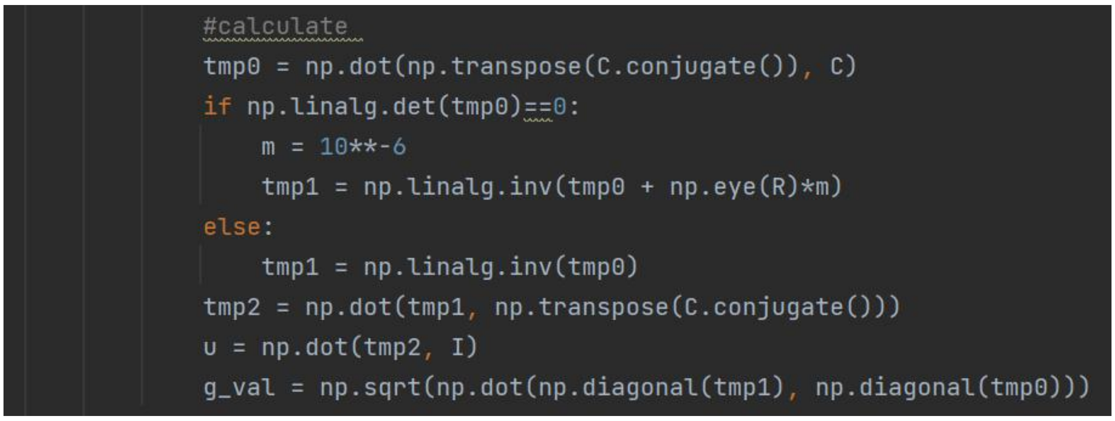
```
n_cov_inv_half = fractional_matrix_power(noise_cov, -1 / 2)
m_w_alias_img = np.zeros_like(m_alias_img) #256×128×8
c_w_coil_sen = np.zeros_like(c_coil_sen) #256×256×8
for i in range(nc):  # number of chanel
    for j in range(nc):
        m_w_alias_img[:, :, i] += n_cov_inv_half[i, j] * m_alias_img[:, :, j]
        c_w_coil_sen[:, :, i] += n_cov_inv_half[i, j] * c_coil_sen[:, :, j]
```

2) create C for each pixel in alias image



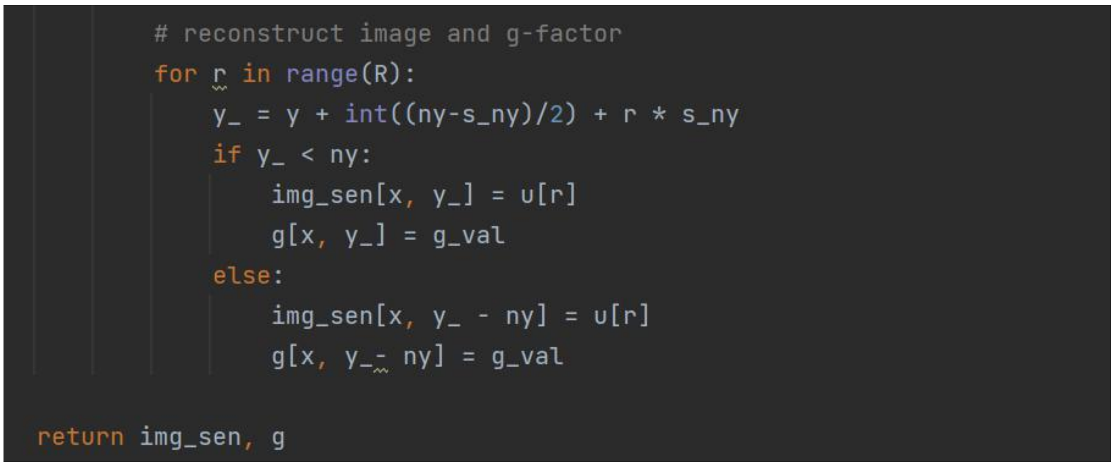
```
for x in range(nx):
    for y in range(s_ny):
        for i in range(nc):
            I[i, 0] = m_w_alias_img[x, y, i]

            # create C for each pixel in alias image
            for r in range(R):
                y_ = y + int((ny-s_ny)/2) + r * s_ny
                if y_ < ny:
                    C[i, r] = c_w_coil_sen[x, y_, i]
                else:
                    C[i, r] = c_w_coil_sen[x, y_-ny, i]
```

3) Calculate the pixel value 'u' in reconstructed MRI image and the g-factor, based on the equation below.



```python
            #calculate
            tmp0 = np.dot(np.transpose(C.conjugate()), C)
            if np.linalg.det(tmp0)==0:
                m = 10**-6
                tmp1 = np.linalg.inv(tmp0 + np.eye(R)*m)
            else:
                tmp1 = np.linalg.inv(tmp0)
            tmp2 = np.dot(tmp1, np.transpose(C.conjugate()))
            u = np.dot(tmp2, I)
            g_val = np.sqrt(np.dot(np.diagonal(tmp1), np.diagonal(tmp0)))
```

4) reconstruct image and g-factor



```python
            # reconstruct image and g-factor
            for r in range(R):
                y_ = y + int((ny-s_ny)/2) + r * s_ny
                if y_ < ny:
                    img_sen[x, y_] = u[r]
                    g[x, y_] = g_val
                else:
                    img_sen[x, y_ - ny] = u[r]
                    g[x, y_- ny] = g_val

    return img_sen, g
```