Chapter-10 Best Approximations

tag

▼ Linear Least Squares Problem

$$(A^T A)x = A^T b$$

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix}$$

Sol:-

$$\underbrace{\begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}_{2\times 3}}_{A^t} \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix}}_{A\ 3\times 2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}_{2\times 3} \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix}_{3\times 1}$$

$$\begin{bmatrix} 5 & 3 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

$$\left[\begin{array}{cc:c} 5 & 3 & 0 \\ 3 & 3 & 6 \end{array}\right] \Rightarrow \left[\begin{array}{cc:c} 5 & 3 & 0 \\ \cancel{3}\,1 & 1 & 2 \end{array}\right]$$

$5R_2 - R_1$

$$\left[\begin{array}{cc:c} 5 & 3 & 0 \\ 0 & 2 & 10 \end{array}\right] \qquad \begin{aligned} 5x_1 + 3x_2 &= 0 \\ 2x_2 &= 10 \\ x_2 &= 5 \end{aligned}$$

$$5x_1 + 3(5) = 0$$

$$5x_1 = -15$$

$$\boxed{x_1 = -3} \quad \boxed{x_2 = 5}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 5 \end{bmatrix}$$

To find Error: $\|b - A\hat{x}\|^2$

$$\begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 5 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

$$\sqrt{(1)^2 + (-2)^2 + (1)^2} = \sqrt{6}.$$

```
import numpy as np
import numpy.linalg as npl

A = np.array([[1.0 , 1.0],
              [-1.0, 1.0],
              [1.0, 1.0]])
b = np.array([[2.0],
              [1.0],
              [3.0]])
(x, residual, rank,s) = npl.lstsq(A,b,rcond=None)
# Note that we use rcond=None which is the new default value since v.1.14.0
print('x= '); print(x) #solution
print('residual= '); print(residual) #error ||Ax-b||
print('rank= '); print(rank) #rank of A
print('singular values= '); print(s) #eigen values of A
```

```
# Compute the Moore-Penrose inverse
tmp = npl.inv(np.dot(np.transpose(A),A))
Amp= np.dot(tmp,np.transpose(A))
```

```
x = np.dot(Amp,b)
print('x= '); print(x)
```

```
print('x= '); print(npl.pinv(A)@b)
```

Performance Comparison

Method	**Speed**	**Numerical Stability**	**Use Case**
`np.linalg.lstsq`	Fastest	Very high	Best for large, well-conditioned least squares problems.
Manual Moore-Penrose	Slowest	Low to Moderate	Good for understanding the method but inefficient in practice.
`np.linalg.pinv`	Moderate	Very high	Best for rank-deficient or ill-conditioned matrices.

▼ Gram-Schmidt Ortho-Normalization

$$A = \begin{bmatrix} 1 & 2 & 2 \\ -1 & 1 & 2 \\ -1 & 0 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

$$u_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \qquad u_2 = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} \qquad u_3 = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 2 \end{bmatrix}$$

$v_1 = u_1 \qquad \|v_1\| = \sqrt{1+1+1+1} = \sqrt{4} = 2$

$v_2 = u_2 - \text{proj}_{v_1} u_2$

$v_3 = u_3 - \text{proj}_{v_1} u_3 - \text{proj}_{v_2} u_3$

$\underline{V2}$

$$v_2 = u_2 - \frac{u_2 v_1}{v_1 v_1} v_1$$

$$u_2 v_1 = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = 2 - \cancel{1} + 0 + \cancel{1} = 2.$$

$$v_1 v_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = 1+1+1+1 = 4$$

$$v_2 = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \frac{\cancel{2}}{\cancel{4}2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1/2 \\ -1/2 \\ -1/2 \\ 1/2 \end{bmatrix}$$

$$v_2 = \begin{bmatrix} 3/2 \\ 3/2 \\ 1/2 \\ +1/2 \end{bmatrix} \qquad \|v_2\| = \sqrt{\frac{9}{4} + \frac{9}{4} + \frac{1}{4} + \frac{1}{4}} = \sqrt{\frac{20}{4}} = \sqrt{5}$$

v_3

$$\text{proj}_{v_1} u_3 = \frac{u_3 v_1}{v_1 v_1} v_1$$

$$u_3 v_1 = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \cancel{2} - \cancel{2} - 1 + 2 = 1$$

$$v_1 v_1 = 4$$

$$\text{proj}_{v_1} u_3 = \begin{bmatrix} 1/4 \\ -1/4 \\ -1/4 \\ 1/4 \end{bmatrix}$$

$$\text{proj}_{v_2} u_3 = \frac{u_3 v_2}{v_2 v_2} v_2$$

$$u_3 v_2 = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 3/2 \\ 3/2 \\ 1/2 \\ 1/2 \end{bmatrix} = \frac{6}{2} + \frac{6}{2} + \frac{1}{2} + \frac{2}{2} = \frac{15}{2} = 7.5$$

$$v_2 v_2 = \frac{9}{4} + \frac{9}{4} + \frac{1}{4} + \frac{1}{4} = 5$$

$$\frac{7.5}{5} = \frac{3}{2}$$

$$\text{proj}_{v_2} u_3 = \begin{bmatrix} 9/4 \\ 9/4 \\ 3/4 \\ 3/4 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 1/4 \\ -1/4 \\ -1/4 \\ 1/4 \end{bmatrix} - \begin{bmatrix} 9/4 \\ 9/4 \\ 3/4 \\ 3/4 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} -1/2 \\ 0 \\ 1/2 \\ 1 \end{bmatrix} \qquad \|v_3\| = \sqrt{\frac{1}{4} + 0 + \frac{1}{4} + 1} = \cancel{1/\sqrt{2}} \cdot \frac{\sqrt{6}}{2}$$

orthonormal vectors · Basis.

$$\left\{ \begin{pmatrix} 1/2 \\ -1/2 \\ -1/2 \\ 1/2 \end{pmatrix}, \begin{pmatrix} 3/2\sqrt{5} \\ 3/2\sqrt{5} \\ 1/2\sqrt{5} \\ 1/2\sqrt{5} \end{pmatrix}, \cancel{\begin{pmatrix} -1/2\sqrt{2} \\ 0 \\ 1/2\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}} \right\} \begin{pmatrix} -\sqrt{6}/6 \\ 0 \\ \sqrt{6}/6 \\ 2/\sqrt{6} \end{pmatrix}$$

```
def gram_schmidt(X):
    E = np.zeros(np.shape(X))
    (m,n)=np.shape(X)
    E[:,0] = X[:,0]/np.sqrt(np.inner(X[:,0],X[:,0]))
    for i in range(1,n):
        E[:,i] = X[:,i]
        for j in range(0,i):
            proj=np.inner(E[:,i],E[:,j])/np.inner(E[:,j],E
[:,j])*E[:,j]
            E[:,i] = E[:,i]-proj
```

```
        E[:,i] = E[:,i]/np.sqrt(np.inner(E[:,i],E[:,i]))
    return E
```

```
import numpy as np
X = np.array([[ 1, 2, 2],
                 [-1, 1, 2],
                 [-1, 0, 1],
                 [ 1, 1, 2]])
E = gram_schmidt(X)
print('E='); print(E)
print('I='); print(np.dot(np.transpose(E),E))
```

▼ Code Explanation

Normalize the First Vector

```
E[:,0] = X[:,0]/np.sqrt(np.inner(X[:,0],X[:,0]))
```

- X[:,0]: The first column of X, representing the first vector.
- $\sqrt{\text{np.inner}(X[:,0], X[:,0])}$: Computes the Euclidean norm of X[:,0].
- E[:,0]: The first orthonormal vector, obtained by dividing X[:,0] by its norm.

Iterate Over Remaining Vectors

```
for i in range(1, n):
    E[:,i] = X[:,i]
```

- Initialize the i-th vector E[:,i] as the i-th column of X.

Project and Subtract Previous Components

```
for j in range(0, i):
    proj = np.inner(E[:,i],E[:,j])/np.inner(E[:,j],E[:,
```

```
j])*E[:,j]
    E[:,i] = E[:,i]-proj
```

- Loop over all previously computed orthonormal vectors E[:,j], where j∈[0,i−1].

Normalize the Resulting Vector

```
E[:,i] = E[:,i]/np.sqrt(np.inner(E[:,i],E[:,i]))
```

Main Code Execution

Verify Orthonormality:

Check if $E^T E = I$(identity matrix):

```
I = np.dot(np.transpose(E), E)
```

This matrix is approximately the identity matrix, confirming E is orthonormal.

▼ QR Factorization

$$A = \begin{array}{ccc} u_1 & u_2 & u_3 \end{array}$$

$$A = \begin{bmatrix} 2 & -2 & 18 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix}$$

Applying Gram Schmidt.

$v_1 = u_1$

$$v_1 = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \qquad \|v_1\| = \sqrt{4+4+1} = \sqrt{9} = 3$$

$$v_2 = u_2 - \text{proj}_{v_1} u_2$$

$$\text{proj}_{v_1} u_2 = \frac{\cancel{}\, u_2 v_1}{v_1 v_1} v_1$$

$$u_2 v_1 = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = -4+2+2 = 0$$

$$v_1 v_1 = 4+4+1 = 9$$

$$\text{proj}_{v_1} u_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$v_2 = \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix} \qquad \|v_2\| = \sqrt{4+1+4} = 3$$

$$v_3 = u_3 - \text{proj}_{v_1} u_3 - \text{proj}_{v_2} u_3$$

$$\text{proj}_{v_1} u_3 = \frac{u_3 v_1}{v_1 v_1} v_1$$

$$u_3 v_1 = \begin{bmatrix} 18 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = 36$$

$$v_1 v_1 = 9$$

$$\frac{36}{9} = \frac{12}{3} = 4$$

$$\text{proj}_{v_1} u_3 = \begin{bmatrix} 8 \\ 8 \\ 4 \end{bmatrix}$$

$$\text{proj}_{v_2} u_3 = \frac{u_3 v_2}{v_2 v_2} v_2$$

$$u_3 v_2 = \begin{bmatrix} 18 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -2 \\ 1 \\ 2 \end{bmatrix} = -36$$

$$\frac{-36}{9} = -4$$

$$v_2 v_2 = 4 + 1 + 4 = 9.$$

$$\text{proj}_{v_2} u_3 = \begin{bmatrix} 8 \\ -4 \\ -8 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} 18 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 8 \\ 8 \\ 4 \end{bmatrix} - \begin{bmatrix} 8 \\ -4 \\ -8 \end{bmatrix}$$

$$v_3 = \begin{bmatrix} 2 \\ -4 \\ 4 \end{bmatrix} \qquad \|v_3\| = \sqrt{4+16+16} = \sqrt{36} = 6$$

$$Q = \begin{bmatrix} 2/3 & -2/3 & 1/3 \\ 2/3 & 1/3 & -2/3 \\ 1/3 & 2/3 & 2/3 \end{bmatrix}$$

$R = Q^T A.$

$$= \begin{bmatrix} 2/3 & 2/3 & 1/3 \\ -2/3 & 1/3 & 2/3 \\ 1/3 & -2/3 & 2/3 \end{bmatrix} \begin{bmatrix} 2 & -2 & 18 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 0 & 12 \\ 0 & 3 & -12 \\ 0 & 0 & 6 \end{bmatrix}$$

$Rx = Q^t b$

for solution.

```
import numpy as np
from numpy.linalg import qr
A = np.array([[2.0, -2.0, 18.0],
              [2.0,  1.0,  0.0],
              [1.0,  2.0,  0.0]])
Q, R = qr( A )
print('R= '); print(R)
print('Q= '); print(Q)
```

▼ Singular Value Decomposition

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{2\times 3}$$

$$\underset{2\times3}{A} = \underset{2\times2}{U}\ \underset{2\times3}{\Sigma}\ \underset{3\times3}{V^t}$$

$$A^tA = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}_{3\times2} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{2\times3}$$

$$A^tA = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{3\times3}$$

Eigen values :-

$$\det \begin{vmatrix} 1-\lambda & 1 & 0 \\ 1 & 1-\lambda & 0 \\ 0 & 0 & 1-\lambda \end{vmatrix} = 0$$

$$(1-\lambda)\left[(1-\lambda)(1-\lambda)-0\right] - 1\left[(1-\lambda)-0\right] = 0$$

$$(1-\lambda)\left[(1-\lambda)^2\right] - \left[(1-\lambda)\right] = 0$$

$$(1-\lambda) = 0$$

$$\boxed{\lambda = 1}$$

$$(1-\lambda)\left[(1-\lambda)-1\right] = 0$$

$$(1-\lambda) = 0 \Rightarrow \boxed{\lambda = 1}$$

$$(1-\lambda) = 1$$

$$1 = 1+\lambda$$

$$\boxed{\lambda = 0}$$

$$(1-\lambda)\left[(1-\lambda)^2 - 1\right] = 0$$

$$\boxed{\lambda = 1}$$

$$(1-\lambda)^2 = 1$$

$$\cancel{1} - 2\lambda + \lambda^2 - \cancel{1} = 0$$

$$\lambda(-2+\lambda) = 0$$

$$\boxed{\lambda = 0}, \quad \boxed{\lambda = 2}$$

$$\lambda = 2, 1, 0$$

Day / Date

For $\lambda = 2$

$$\left[\begin{array}{ccc:c} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{array}\right]$$

$R_2 + R_1$

$$\left[\begin{array}{ccc:c} -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{array}\right]$$

$-x_1 + x_2 = 0 \Rightarrow x_1 = +x_2$

$x_3 = 0$

x_2 is free.

$$v_1 = \begin{pmatrix} +1 \\ 1 \\ 0 \end{pmatrix} \qquad \|v_1\| = \sqrt{2}$$

For $\lambda = 1$

$$\left[\begin{array}{ccc:c} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

$x_2 = 0$

$x_1 = 0$

x_3 is free

$$v_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad \|v_2\| = 1 \text{ alr orthonormal}$$

For $\lambda = 0$

$$\left[\begin{array}{ccc:c} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

$x_1 + x_2 = 0 \Rightarrow x_1 = -x_2$

$x_3 = 0$

x_2 is free.

$$v_3 = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \qquad \|v_3\| = \sqrt{2}$$

$$V = \begin{array}{c} \begin{pmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 0 & 1 & 0 \end{pmatrix} \\ \begin{array}{ccc} v_1 & \quad v_2 & \quad v_3 \end{array} \end{array}$$

Now, $u_1 = \dfrac{Av_1}{\sigma_1} \qquad \sigma_1 = \sqrt{2}$

Prince

Day / Date

$$Av_1 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{2\times 3} \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix}_{3\times 1}$$

$$Av_1 = \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix}$$

$$u_1 = \begin{bmatrix} \sqrt{2}/\sqrt{2} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\Rightarrow u_2 = \frac{Av_2}{\sigma_2}, \quad \sigma_2 = 1$$

$$Av_2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{2\times 3} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_{3\times 1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$u_2 = \frac{Av_2}{\sigma_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\Rightarrow u_3 = \frac{Av_3}{\sigma_3}, \quad \sigma_3 = 0 \quad = \cancel{\frac{Av_3}{0}}$$

$$\cancel{Av_3 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} \\ -1/\sqrt{2} \\ 0 \end{bmatrix} = \begin{bmatrix} -\sqrt{2} \\ 0 \end{bmatrix}}$$

$$U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{2\times 2}$$

$$\Sigma = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}_{2\times 2} \begin{pmatrix} \sqrt{2} & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}_{2\times 3} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \end{pmatrix}$$

Prince

```
import numpy.linalg as npl

A = np.array([[1.0, 2.0, 3.0],
              [3.0, 2.0, 1.0]])
U, S, V = npl.svd( A )
print(S)
print(U)
print(V)
```

```
# Conver matrix S into rectangular matrix
Sigma = np.zeros((A.shape[0], A.shape[1]))
Sigma[:A.shape[0], :A.shape[0]]=np.diag(S)

print(U@Sigma@V)
```

▼ Application in Image Processing

```
import matplotlib.pyplot as plt
import imageio.v2 as imageio
import numpy as np

photo = imageio.imread("Newton.jpg")/255; # Read the image
into the array photo
print(photo.shape)
plt.imshow(photo) # Plot the image on the screen
plt.show()

row, col, dim = photo.shape
```

```
Red = np.zeros(photo.shape)
Green = np.zeros(photo.shape)
```

```
Blue = np.zeros(photo.shape)
# Plot the different matrices using imshow
f, axs = plt.subplots(2,2,figsize=(15,15))
# Separate the three basic colors
Red[:,:,0] = photo[:,:,0]; Green[:,:,1] = photo[:,:,1];
Blue[:,:,2] = photo[:,:,2]
plt.subplot(2,2,1); plt.imshow(Red); plt.subplot(2,2,2); pl
t.imshow(Green)
plt.subplot(2,2,3); plt.imshow(Blue); plt.subplot(2,2,4); p
lt.imshow(photo)
plt.show()
```

```
Red = photo[:,:,0]
Green = photo[:,:,1]
Blue = photo[:,:,2]

U_r,S_r,V_r = npl.svd(Red)
U_g,S_g,V_g = npl.svd(Green)
U_b,S_b,V_b = npl.svd(Blue)

sequence = [5, 10, 20, 40, 100, 400]

f, axs = plt.subplots(2,3,figsize=(15,15))

j=0
for k in sequence:
    U_r_c = U_r[:,0:k]
    V_r_c = V_r[0:k,:]
    U_g_c = U_g[:,0:k]
    V_g_c = V_g[0:k,:]
    U_b_c = U_b[:,0:k]
    V_b_c = V_b[0:k,:]
    S_r_c = np.diag(S_r[0:k])
    S_g_c = np.diag(S_g[0:k])
    S_b_c = np.diag(S_b[0:k])
```

```
    comp_img_r = np.dot(U_r_c, np.dot(S_r_c,V_r_c))
    comp_img_g = np.dot(U_g_c, np.dot(S_g_c,V_g_c))
    comp_img_b = np.dot(U_b_c, np.dot(S_b_c,V_b_c))
    comp_img = np.zeros((row, col, 3))
    comp_img[:,:,0] = comp_img_r
    comp_img[:,:,1] = comp_img_g
    comp_img[:,:,2] = comp_img_b
    comp_img[comp_img < 0] = 0
    comp_img[comp_img > 1] = 1
    j=j+1
    plt.subplot(2,3,j)
    plt.title('Rank %d'%(k))
    plt.imshow(comp_img)

plt.show()
```