

Name: ZHANG XINJIE
Student ID Number: 20M14457
E-mail: zhang.x.a2@m.titech.ac.jp

Ex.2 エントロピーの計算

課題：読み込んだ画像(pepper.png)を情報源とみなし、画素値(0から255の整数値)をシンボルとみなしたときのエントロピーを計算せよ。このとき、各シンボルの発生確率は生起頻度を正規化することで得よ。答えは約7.6 [bit]となるはずである。

The entropy of an image can be obtained by calculating the entropy of the pixel-values at each pixel position (i,j) , within a 2-D region centered at (i,j) . In this case, the region size is configured to be 256×256 .

To calculate the entropy of a memoryless information source $S = \{s_1, s_2, \dots, s_N\}$ with N symbols, given the corresponding probability of each symbol p_1, p_2, \dots, p_N , the entropy of the information source can be obtained by:

$$H(S) = \sum_{i=1}^N p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^N p_i \log_2 p_i$$

Step 1: Read the image, and transform the image into 2-D array.

```
In [21]: !git clone https://github.com/mdipcit/standard_images/

Cloning into 'standard_images'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (10/10), done.

In [22]: cd standard_images/

/content/standard_images/standard_images

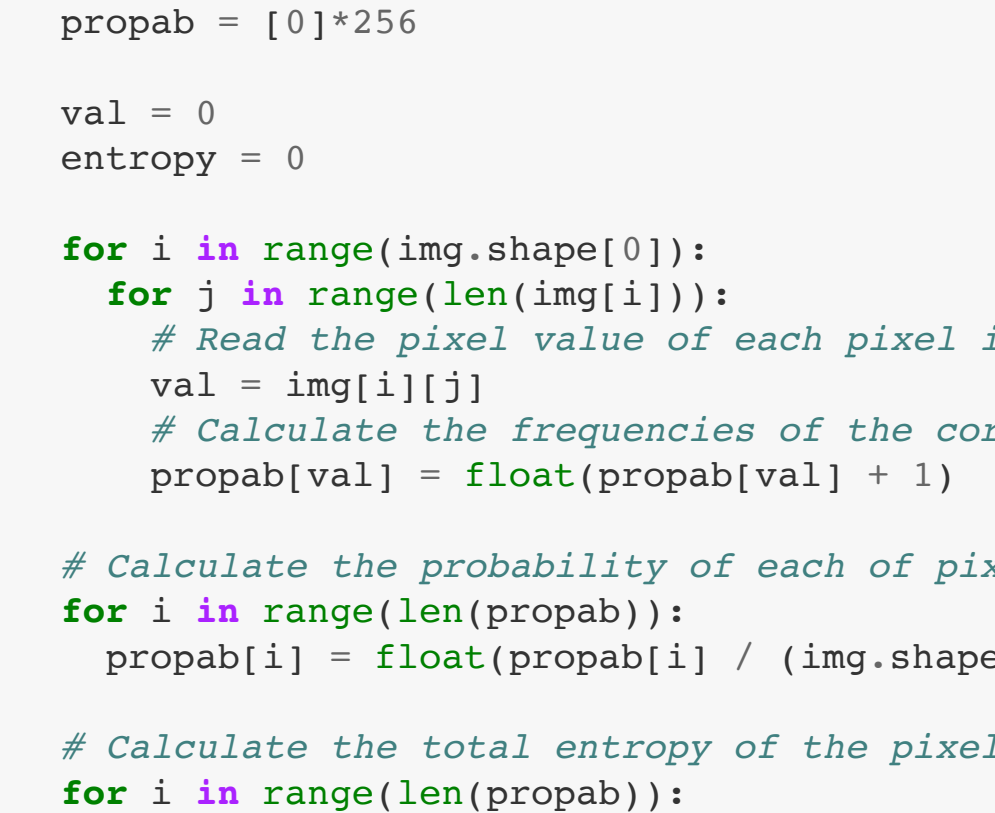
In [3]: import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

In [13]: im = Image.open('pepper.png')
im = np.asarray(im)
print(im.shape[0])
print(im)
```

```
256
[[ 21  43  45 ...  52  49  56]
 [ 27 109 105 ... 188 180 180]
 [ 27 120 106 ... 186 183 180]
 ...
 [22 129 100 ... 197 201 202]
 [ 20 138 109 ... 207 203 193]
 [ 20 113 112 ... 208 194 197]]
```

Show the histogram of pixel values of this image.

```
In [5]: hist, bins = np.histogram(im.ravel(), 256, [0,256])
plt.hist(im.ravel(),256,[0,256])
plt.show()
```



Step 2: Calculate the probability of each of the pixel values, and calculate the entropy using the formula above.

```
In [18]: def calcEntropy(img):
# Initialize a 1-D list to store the probability with length of the number of pixel values.
propab = [0]*256

val = 0
entropy = 0

for i in range(img.shape[0]):
    for j in range(len(img[i])):
        # Read the pixel value of each pixel in the image.
        val = img[i][j]
        # Calculate the frequencies of the corresponding pixel values.
        propab[val] = float(propab[val] + 1)

# Calculate the probability of each of pixel values.
for i in range(len(propab)):
    propab[i] = float(propab[i] / (img.shape[0]*img.shape[1]))

# Calculate the total entropy of the pixels in the image.
for i in range(len(propab)):
    if(propab[i] != 0):
        entropy = entropy
    else:
        entropy = float(entropy + propab[i]*(np.log2(propab[i])))

return entropy

In [23]: entropy_im = calcEntropy(im)
print("The entropy of pepper.png is", entropy_im)

The entropy of pepper.png is 7.633479825985054
```

演習 4 KLTの基底の計算

課題

配布資料6のモデル化された自己共分散行列 C_{model} を生成し（ 8×8 行列。 ρ はパラメータとする）、このときのKLTの基底ベクトルを求め可視化せよ。また、 ρ を変化させたときに基底がどのように変化するかを観察せよ。余裕があれば、得られたKLTの基底を1次元8点DCTの基底と比較せよ。

ヒント

- 固有ベクトルの計算には`np.linalg.eig`を使用するとよい。
- 1次元8点DCTの基底は ρ が1以下であれば可視化できる。`plt.imshow(det(np.eye(8,8)))`
- `toeplitz`行列は`scipy.linalg.toeplitz`を使うと簡単に作れる（使わなくても可き）

In image coding, orthogonal transformation is used to remove the inter-pixel correlation. The Mean Squared Error (MSE) $E_m[d^{(m)}$] of transformation coefficients $y^{(m)}$ in all blocks is the target to minimize before and after the orthogonal transformation. The average bit length of the quantization for $y^{(m)}$ in each block is R . The minimization of MSE can be formulated as the following:

$$\mathbf{r}^* = \arg \min_{\mathbf{r}} MSE(\arg \min_{\mathbf{r}} E_m[d^{(m)}]) \quad \text{subject to } R = \frac{1}{64} \sum_{m=1}^{64} r_n$$

The minimum of MSE is

$$MSE_{min} = 2^{-2R} \left[\prod_{n=1}^{64} \sigma_{y_n}^2 \right]^{\frac{1}{64}}$$

Define a autocovariance matrix C_{xx} that:

$$C_{xx} = E_m[\mathbf{x}^{(m)}(\mathbf{x}^{(m)})^T]$$

Karhunen-Loeve transform is the method that uses the eigenvectors of C_{xx} as the basis vectors of autocovariance matrix C_{yy} of coefficient $y^{(m)}$ after transform to remove the inter-pixel correlation.

The flow of KLT

Step 1: To obtain a real symmetric autocovariance matrix, all the eigenvalues λ_i should be real numbers, and the eigenvectors should be corresponding to the eigenvalues that are in descending order.

Step 2: Arranging the eigenvectors in columns to form a 64×64 matrix V .

$$V^T C_{xx} V = V^T E_m[\mathbf{x}^{(m)}(\mathbf{x}^{(m)})^T] V = C_{yy}$$

Step 3: The autocovariance matrix of the transformation coefficient $y^{(m)}$ can be obtained by the transformation $y^{(m)} = V^T x^{(m)}$. The elements $y_i^{(m)}$ is independent from each other.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import toeplitz
from scipy.fft import dct

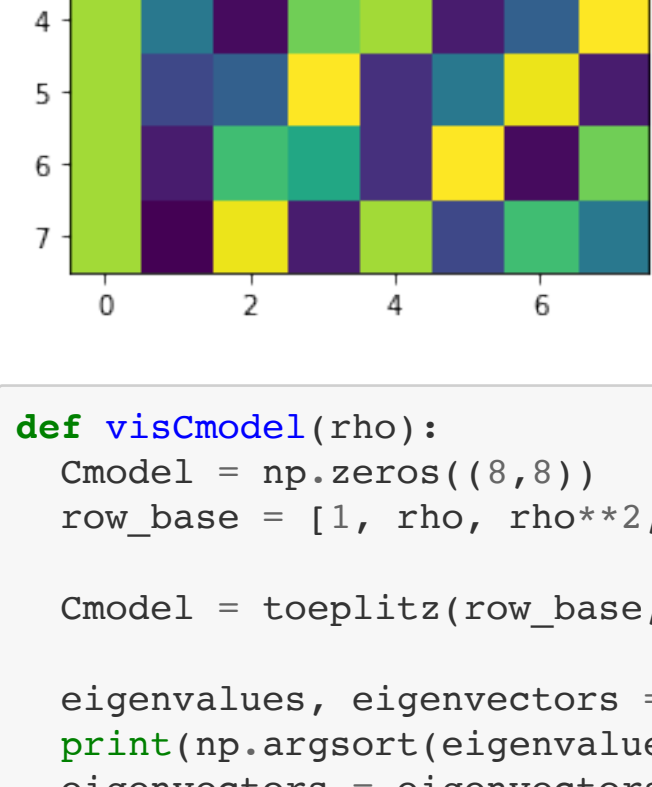
rho=0.99

In [13]: # Create a autocovariance matrix of a 8x8 block.
Cmodel = np.zeros((8,8))
row_base = [1, rho, rho**2, rho**3, rho**4, rho**5, rho**6, rho**7]
Cmodel = toeplitz(row_base, row_base)

# Calculate the eigenvalues and eigenvectors of the matrix, and arrange
# the eigenvectors in the descending order corresponding to the eigenvalues.
eigenvalues, eigenvectors = np.linalg.eig(Cmodel)
print(np.argsort(eigenvalues))
eigenvectors = eigenvectors[:,np.argsort(eigenvalues)[::-1]]
plt.imshow(eigenvectors)
```

```
[7 6 5 4 3 2 1 0]
```

Out[13]: <matplotlib.image.AxesImage at 0x7ff24eba9a58>

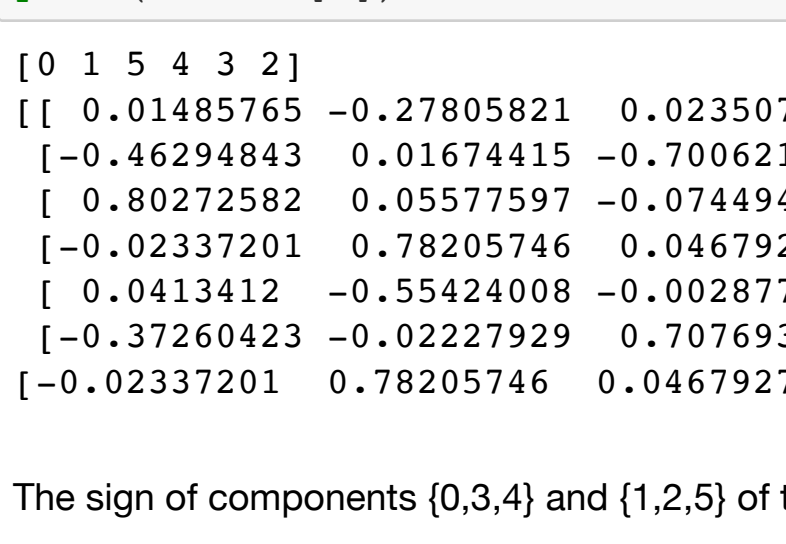
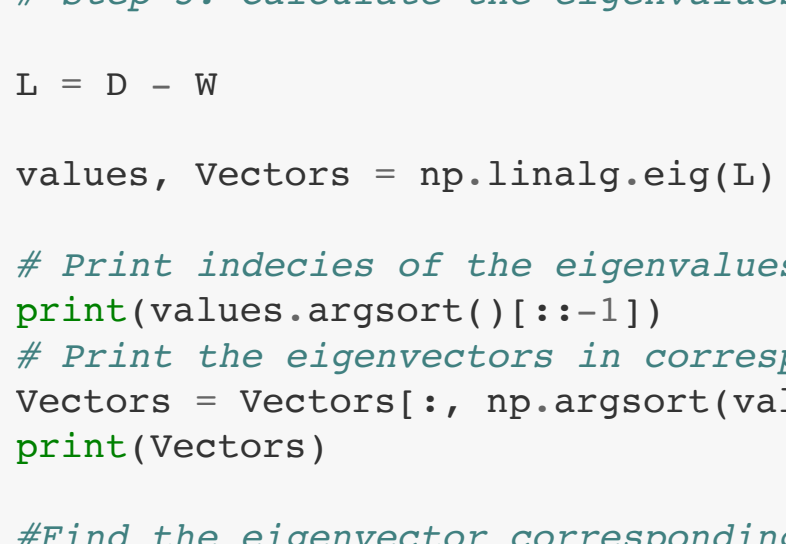
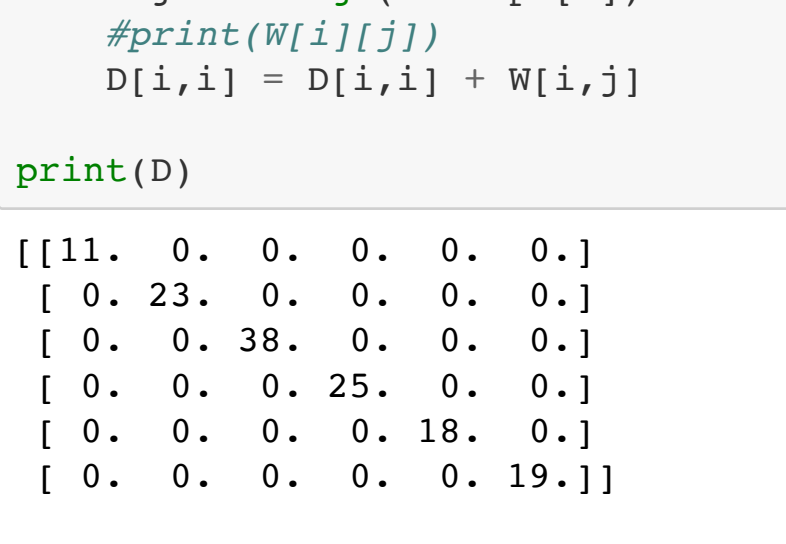
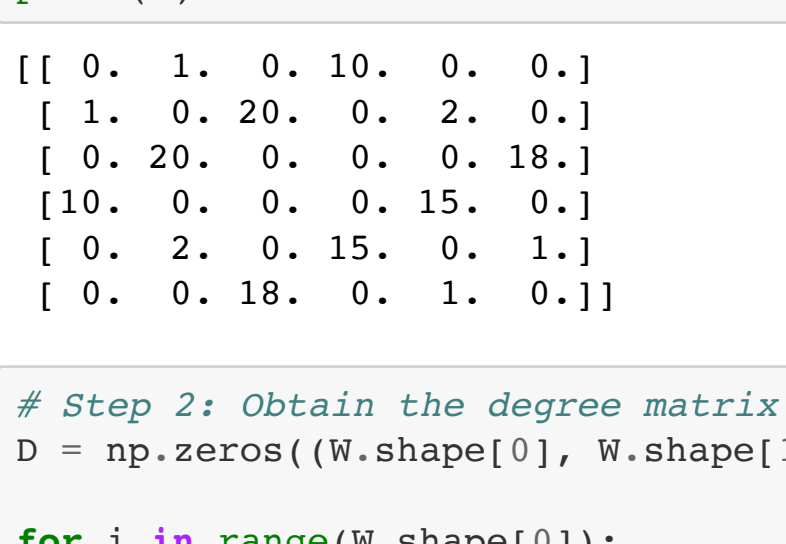
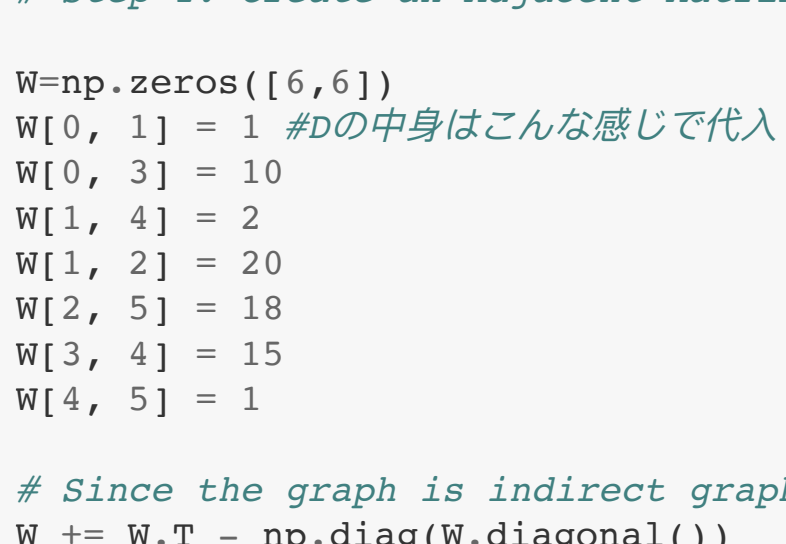
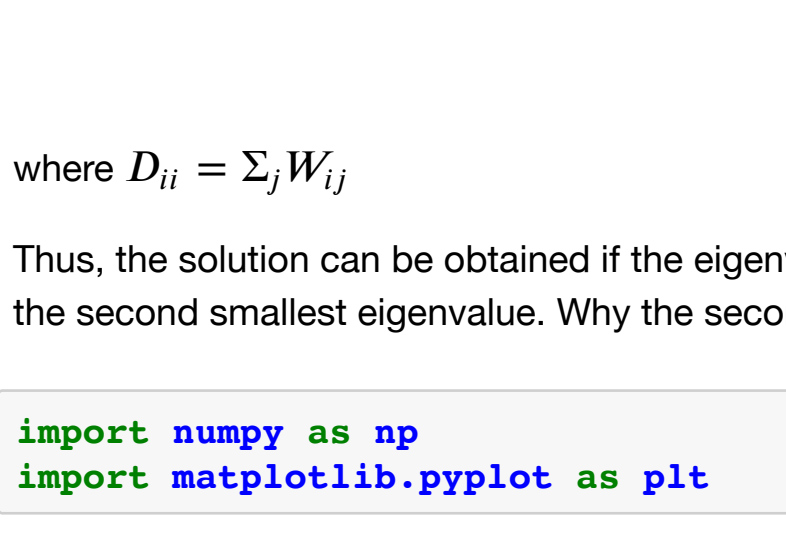
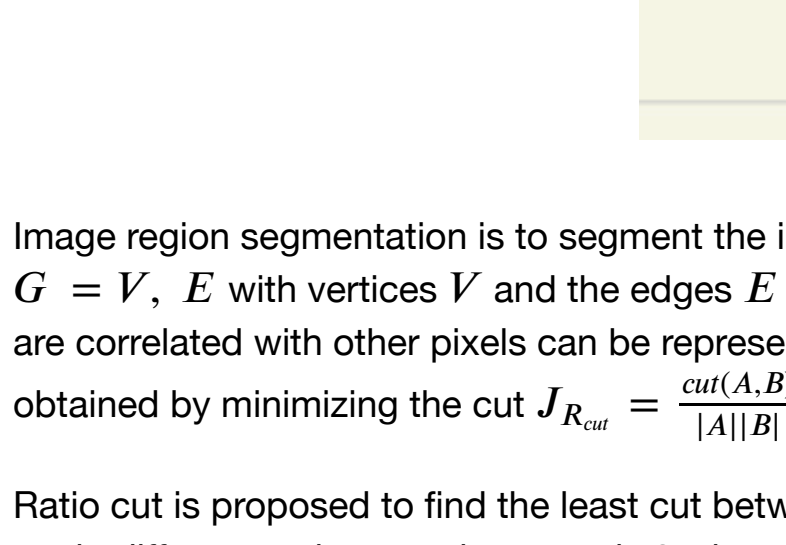
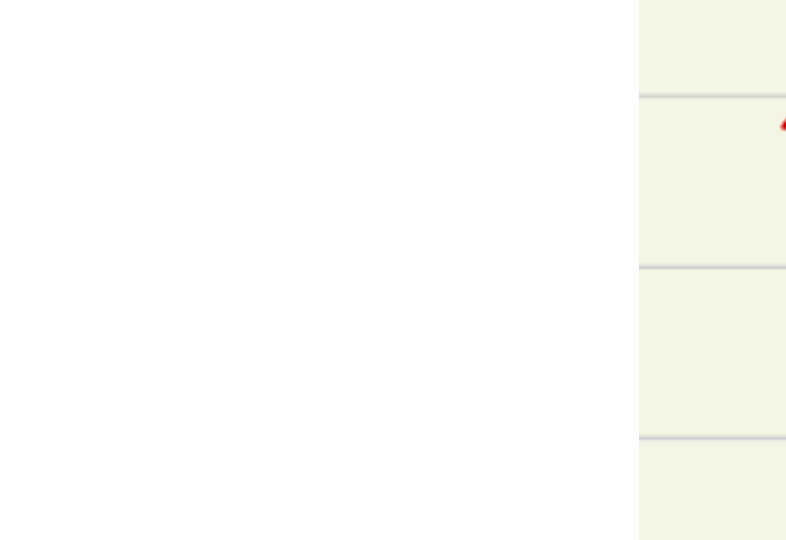
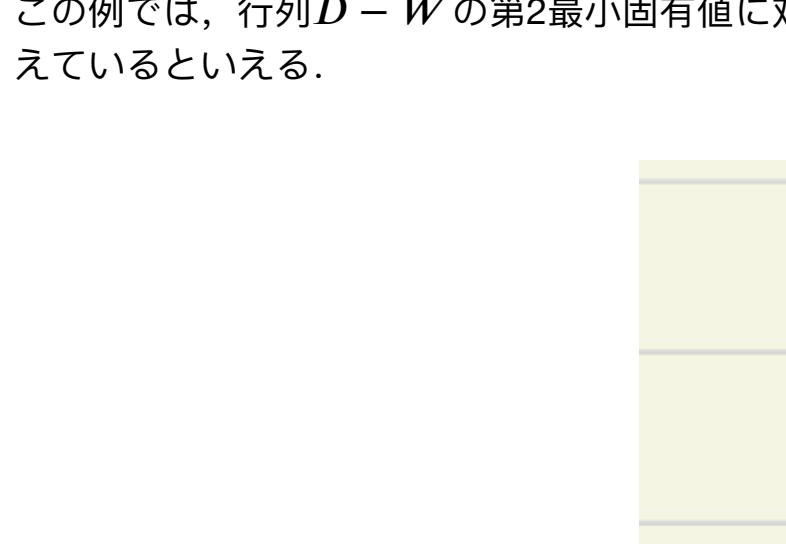
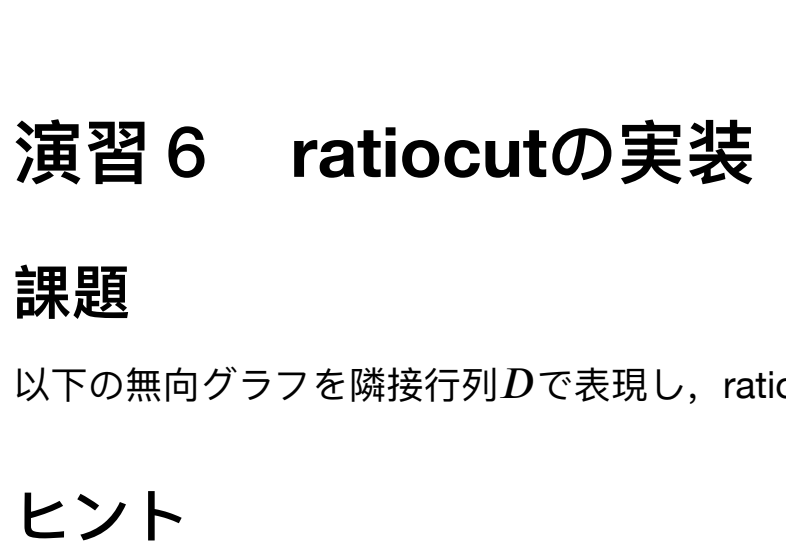
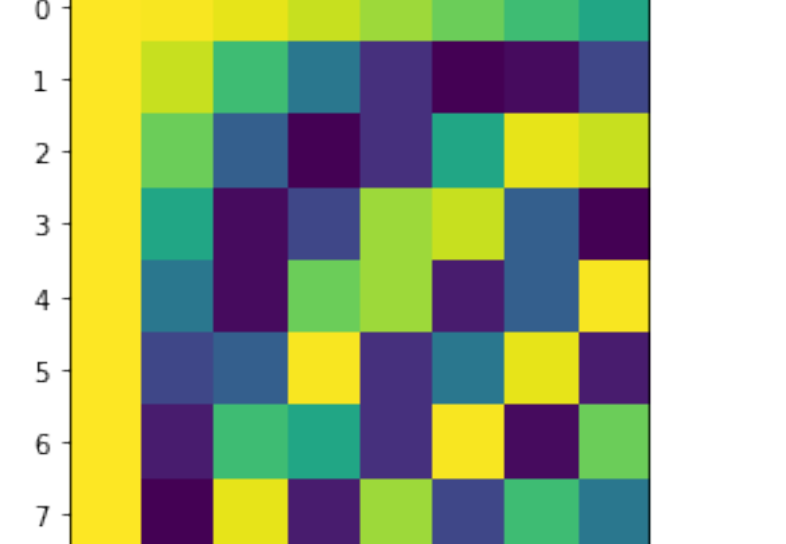
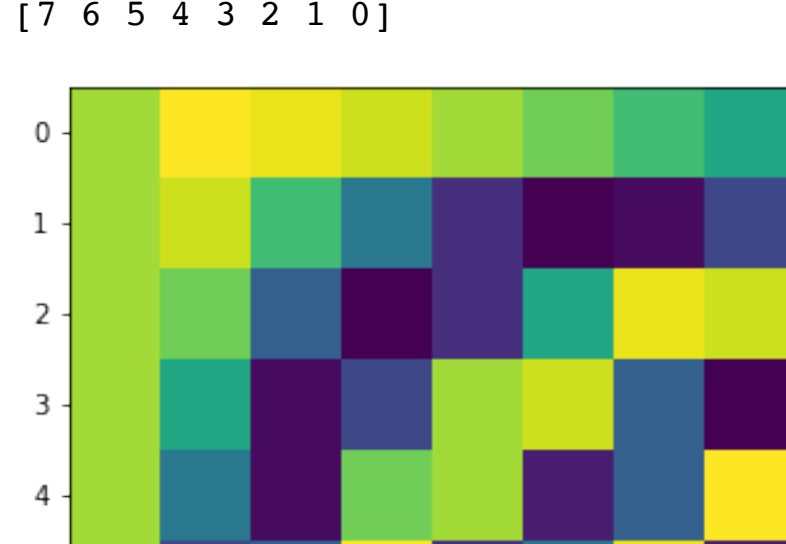
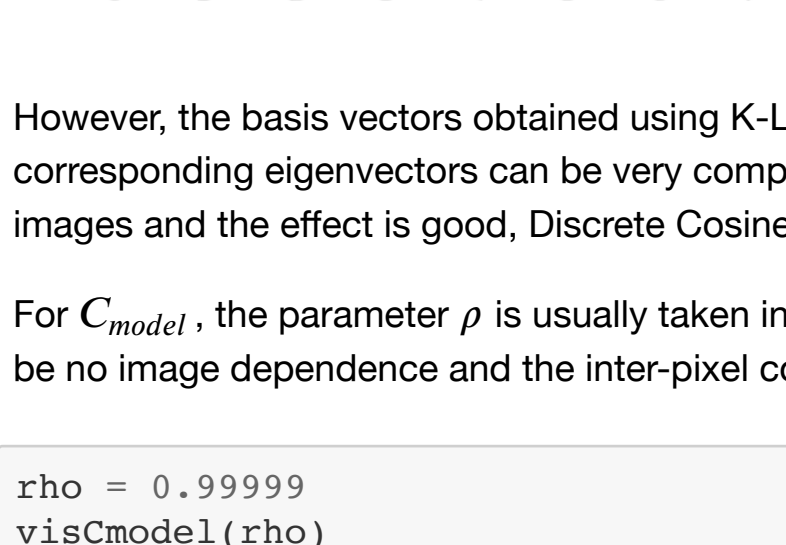
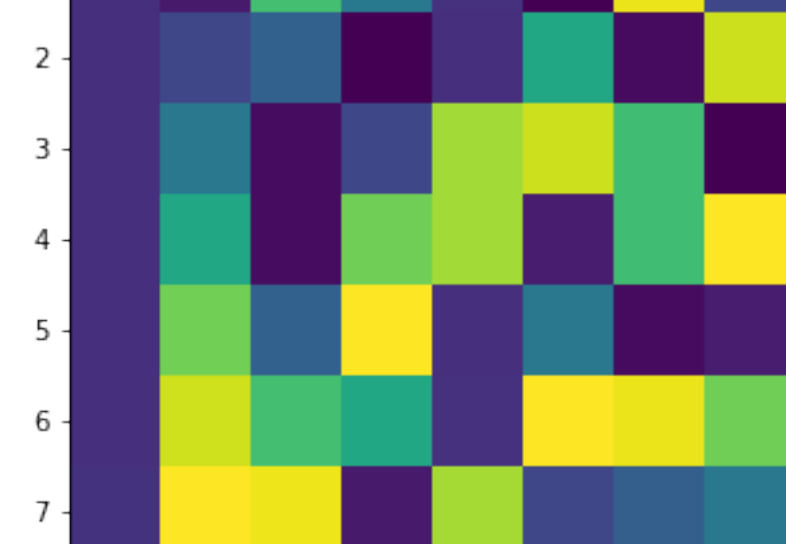
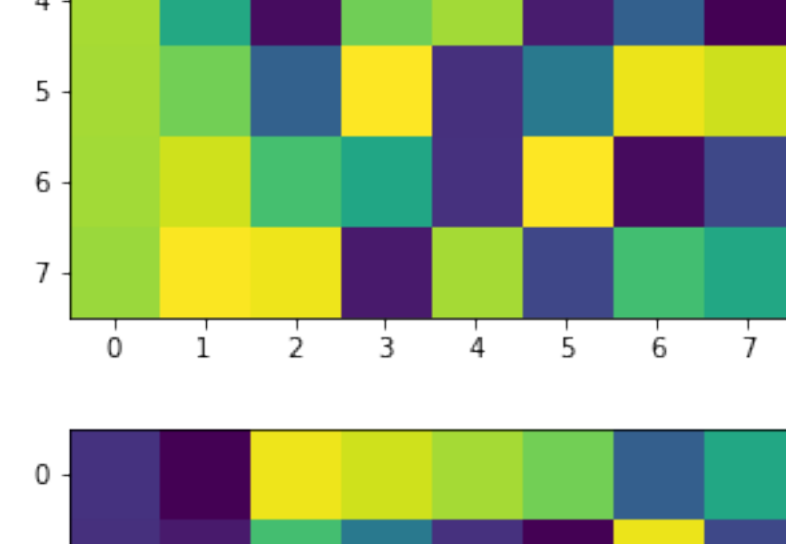
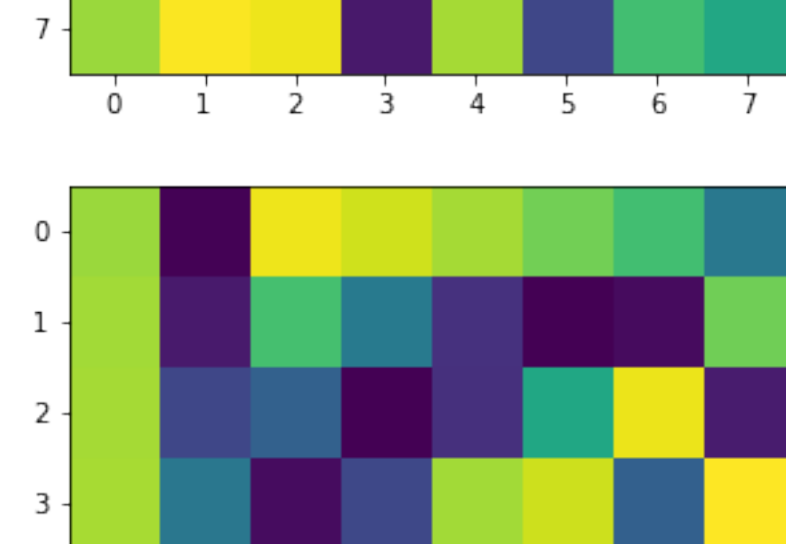
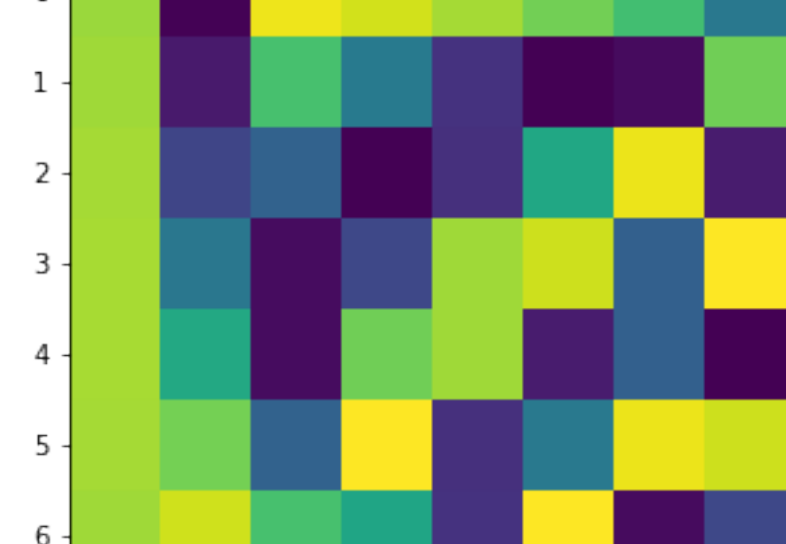
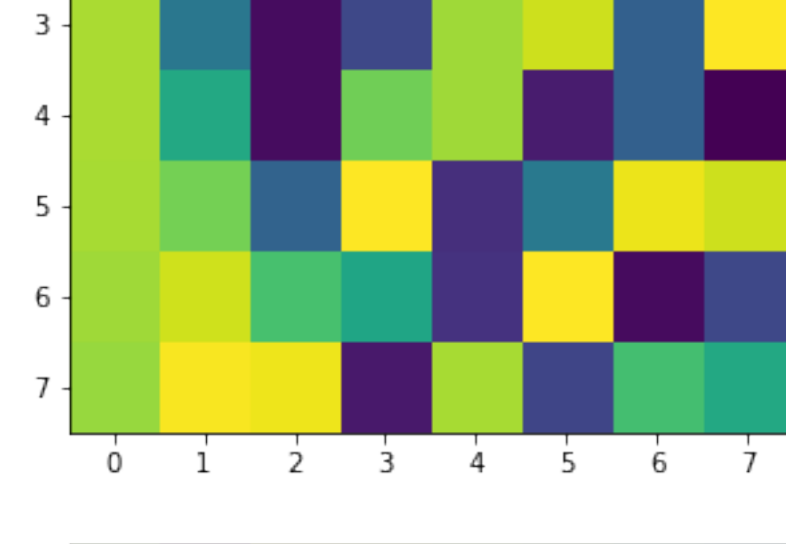
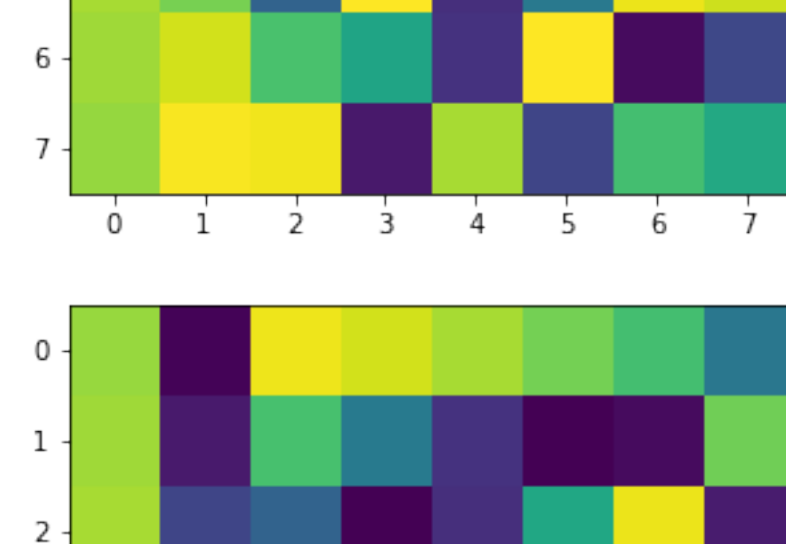
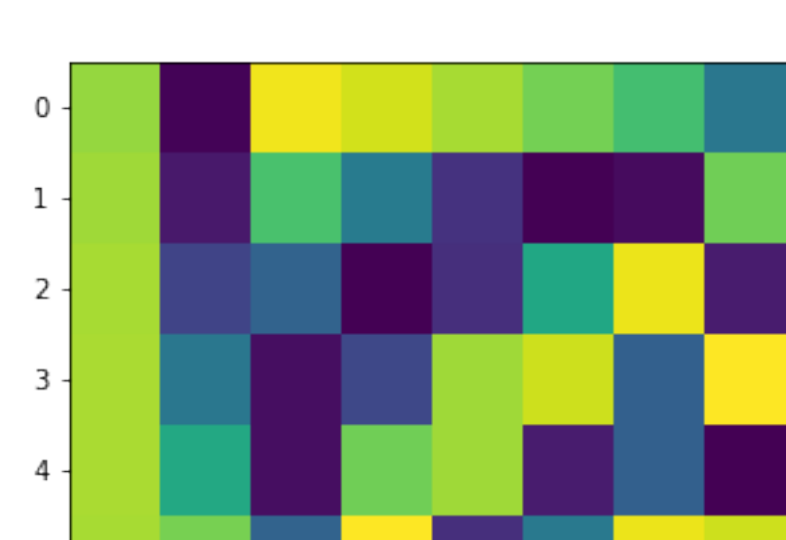
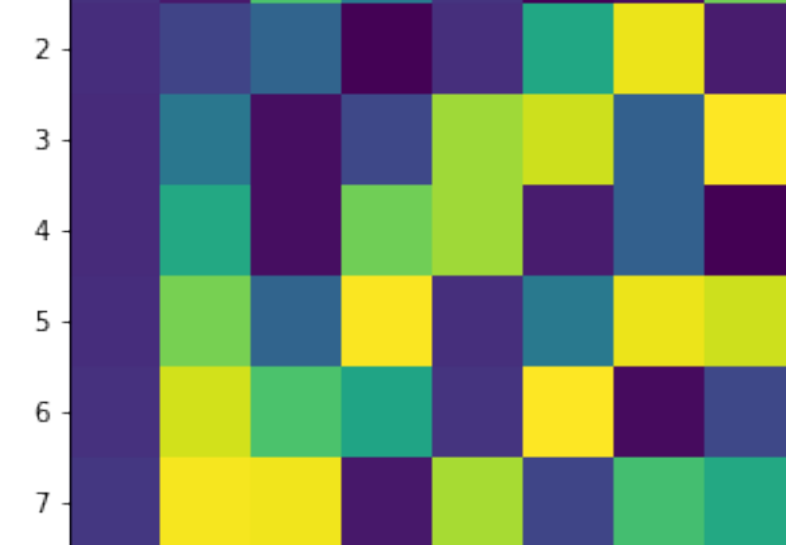
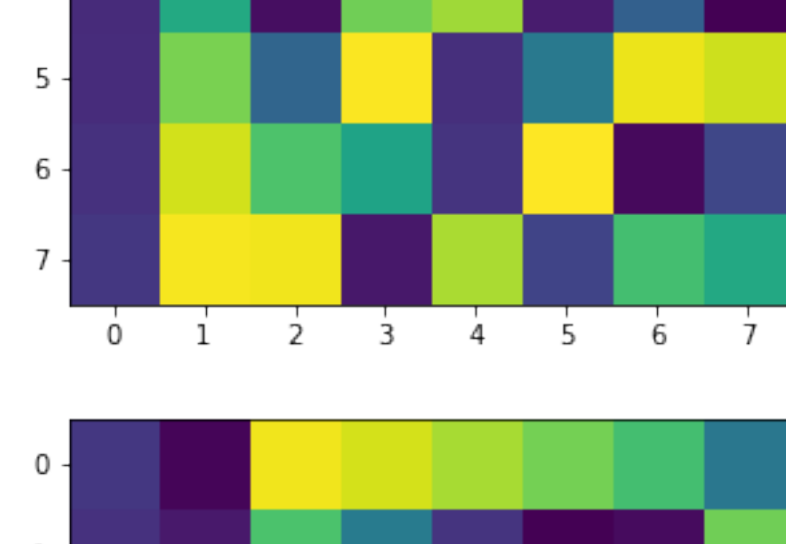
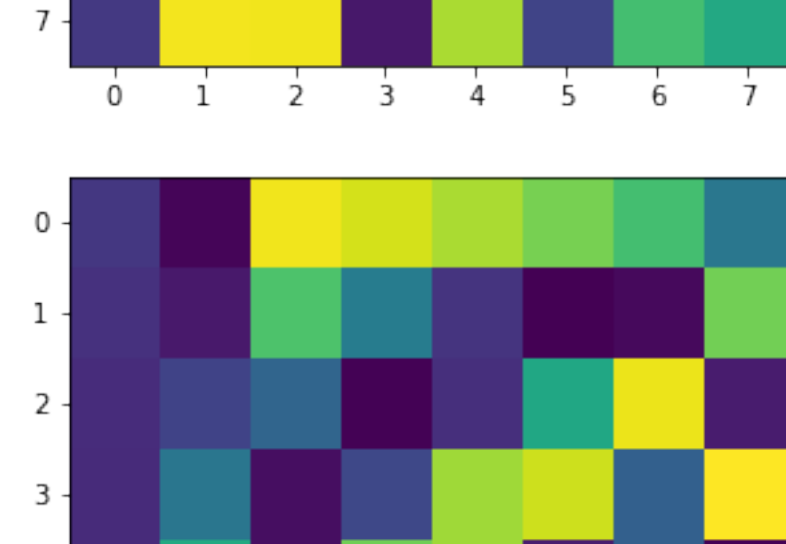
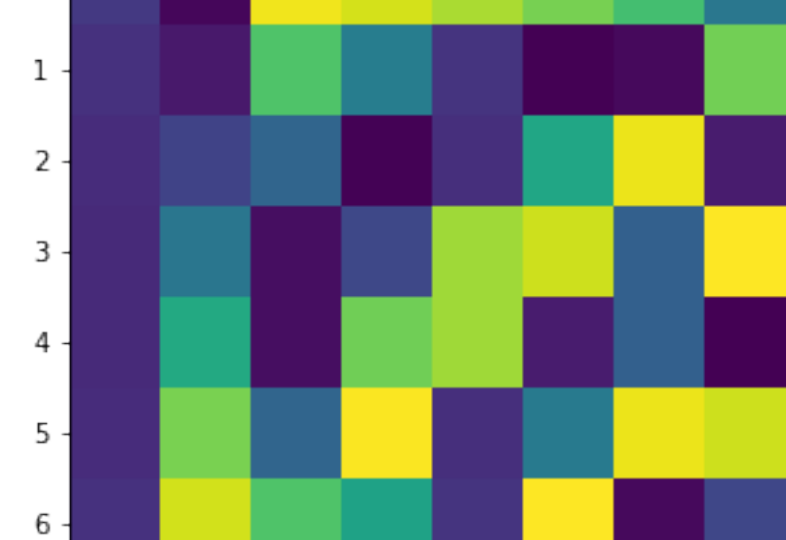
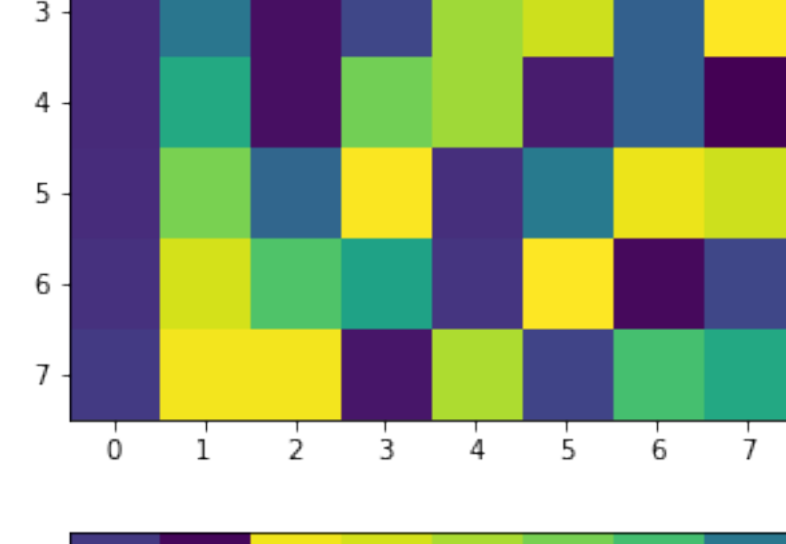


```
In [14]: def visCmodel(rho):
Cmodel = np.zeros((8,8))
row_base = [1, rho, rho**2, rho**3, rho**4, rho**5, rho**6, rho**7]

Cmodel = toeplitz(row_base, row_base)

eigenvalues, eigenvectors = np.linalg.eig(Cmodel)
print(np.argsort(eigenvalues))
eigenvectors = eigenvectors[:,np.argsort(eigenvalues)[::-1]]
plt.figure(figsize=(5, 5))
plt.imshow(eigenvectors)

# Change the rho in the range (0.90, 0.98)
rho_value = [0.90, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98]
for rho in rho_value:
    visCmodel(rho)
```



However, the basis vectors obtained using K-L transform relies on the input signal (image). Finding the autocovariance matrix for each input image and the corresponding eigenvectors can be very computationally inefficient. To find an orthogonal transform that can meet the demand that can be applied for all images and the effect is good, Discrete Cosine Transform (DCT) is proposed.

For C_{model} , the parameter ρ is usually taken in the range of [0.90, 0.98] in a practical image. Considering the limit of K-L transform when $\rho \rightarrow 1$, there would be no image dependence and the inter-pixel correlation be removed.

```
In [11]: rho = 0.99999
visCmodel(rho)

[7 6 5 4 3 2 1 0]
```

```
In [12]: base = dct(np.eye(8,8))
plt.imshow(base)

Out[12]: <matplotlib.image.AxesImage at 0x7ff24f16f7b8>
```

演習 6 ratiocutの実装

課題

以下の無向グラフを隣接行列 D で表現し、`ratiocut`を実行して領域分割を行え。黒字は頂点の番号、赤字は辺の重みを表している。

ヒント

この例では、行列 $D - W$ の第 i 最小固有値に対応する固有ベクトルの成分の符号(sign)が成分 $\{0, 3, 4\}$ と $\{1, 2, 5\}$ で異なっていれば、正しくグラフカットが行えているといえる。

Image region segmentation is to segment the image into several regions that contains meaningful contents. The pixels in the image can be considered as graph $G = V, E$ with vertices V and the edges E connecting neighboring pixels. The weight of the edge connecting vertex i and vertex j is w_{ij} . How the pixels are correlated with other pixels can be represented using degree of similarity, and thus, the image can be segmented in this way. The best segmentation can be obtained by minimizing the cut $J_{R_n} = \frac{cut(A,B)}{|A||B|}$ between regions A and B .

Ratio cut is proposed to find the least cut between two different regions of vertices. Set $|V| = n$, $|A| = a$, $|B| = bn$, $a + b = 1$, $q_i = q_j$ is 1 when i and j are in different regions, and $q_i = q_j$ is 0 when they are in the same regions.

$$J_{R_n} = \frac{cut(A,B)}{|A||B|} = \frac{1}{n} \sum_{i,j} W_{ij} (q_i - q_j)^2 = \frac{1}{n} \frac{\mathbf{q}^T (D - W) \mathbf{q}}{\mathbf{q}^T \mathbf{q}}$$

where $D_{ii} = \sum_j W_{ij}$

Thus, the solution can be obtained if the eigenvectors of symmetric matrix $(D - W)$ can be obtained. The solution would be the eigenvector corresponding to the second smallest eigenvalue. Why the second smallest? Because the smallest eigenvalue probably can be 0.

```
In [16]: import numpy as np
import matplotlib.pyplot as plt

In [18]: # Step 1: Create an Adjacent Matrix

W=np.zeros([6,6])
W[0, 1] = 1 #Dの中身はこんな感じで代入していけばよい
W[0, 3] = 10
W[1, 4] = 2
W[1, 2] = 20
W[2, 5] = 18
W[3, 4] = 15
W[4, 5] = 1

# Since the graph is undirect graph, the adjacent matrix should be symmetric.
W += W.T - np.diag(W.diagonal())
print(W)
```

```
[[ 0.  1.  0. 10.  0.  0.]
 [ 1.  0. 20.  0.  2.  0.]
 [ 0. 20.  0.  0.  0. 18.]
 [10.  0.  0.  0. 15.  0.]
 [ 0.  2.  0. 15.  0.  1.]
 [ 0.  0. 18.  0.  1.  0.]]
```

```
In [19]: # Step 2: Obtain the degree matrix (diagonal)
D = np.zeros((W.shape[0], W.shape[1]))

for i in range(W.shape[0]):
    for j in range(W.shape[1]):
        #print(W[i][j])
        D[i,i] = D[i,i] + W[i,j]

print(D)
```

```
[[11.  0.  0.  0.  0.  0.]
 [ 0. 23.  0.  0.  0.  0.]
 [ 0.  0. 39.  0.  0.  0.]
 [ 0.  0.  0. 25.  0.  0.]
 [ 0.  0.  0.  0. 18.  0.]
 [ 0.  0.  0.  0. 15.  0.]]
```

```
In [38]: # Step 3: Calculate the eigenvalues and eigenvectors of the Laplacian Matrix

L = D - W

values, Vectors = np.linalg.eig(L)

# Print indices of the eigenvalues that are arranged in descending order.
print(values.argsort()[::-1])
# Print the eigenvectors in corresponding order.
Vectors = Vectors[:, np.argsort(values)[::-1]]
print(Vectors)

#Find the eigenvector corresponding to the second smallest eigenvalue
print(Vectors[3])
```

```
[0 1 5 4 3 2]
[[-0.01485765 -0.27865821  0.02350724  0.74000284 -0.45567469  0.40824829]
 [-0.45294843  0.01674415 -0.70062125  0.03192608  0.35614928  0.40824829]
 [ 0.80272582  0.05577597 -0.07449441  0.04216245  0.42252402  0.40824829]
 [-0.02337201  0.78205746  0.04679275 -0.19447483 -0.42563269  0.40824829]
 [ 0.6413412  -0.5424608  -0.00287177 -0.64133207 -0.33634368  0.40824829]
 [-0.37260423 -0.02277929  0.70769345  0.02171552  0.43897776  0.40824829]
 [-0.02337201  0.78205746  0.04679275 -0.19447483 -0.42563269  0.40824829]]
```

The sign of components {0,3,4} and {1,2,5} of the eigenvector corresponding to the second smallest eigenvalue is different. The graph cut is correct.