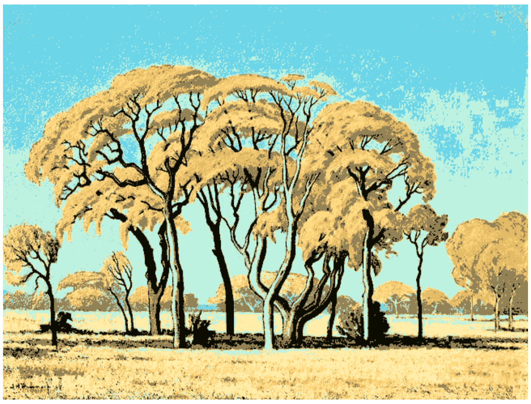
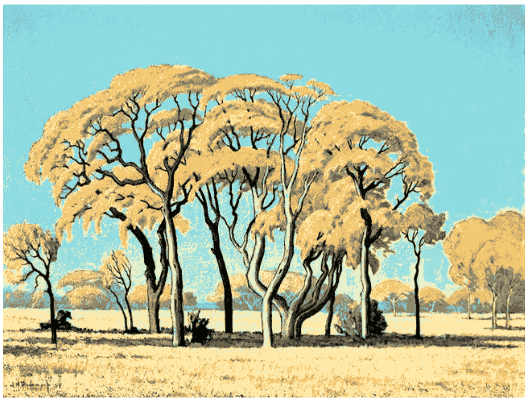
Homework 8

1. **Image 1 ie trees ie 2cf92c40c3f3306321d789f7e9c12893.jpg - segmented into 10, 20 and 50 segments**

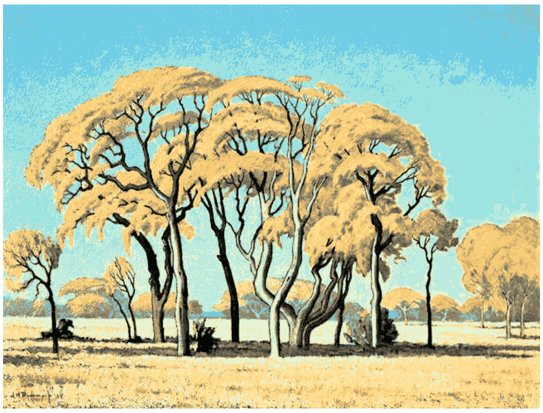
(1) **10**



(2) **20**

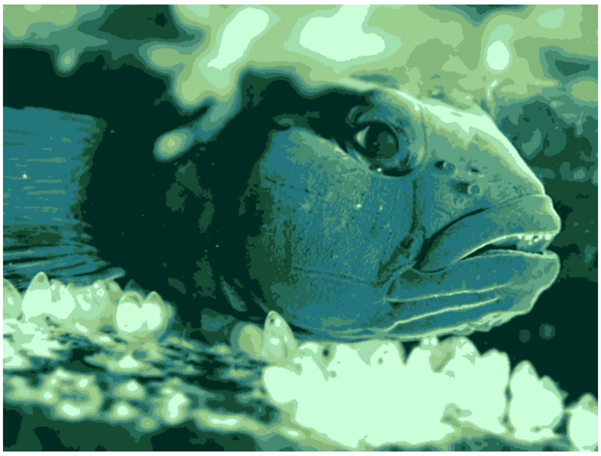


(3) **50**

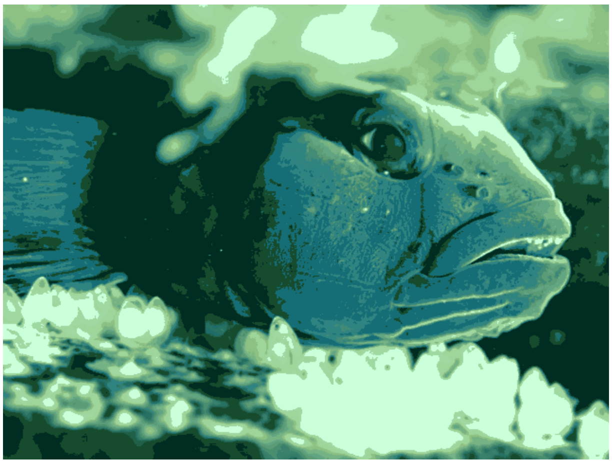


2. **Image 2 ie RobertMixed03.jpg segmented into 10,20,50 segments**

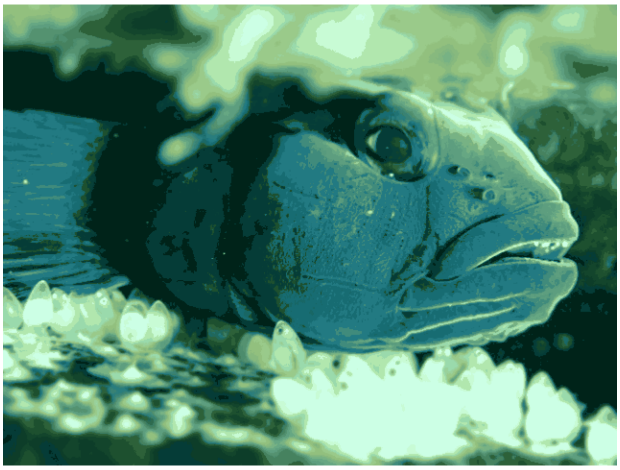
(1) **10**



(2) **20**

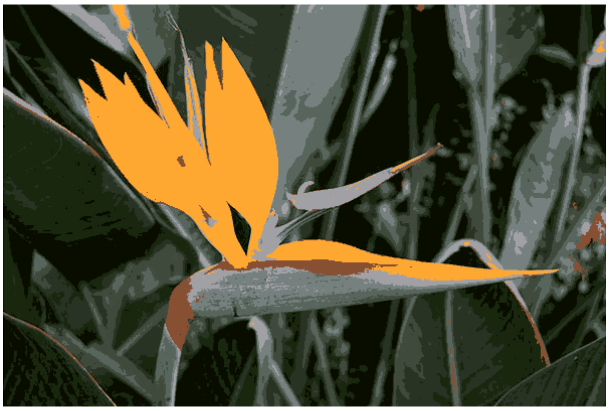


(3) **50**

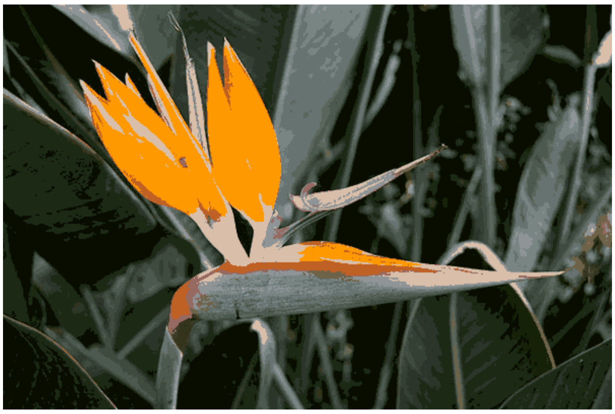


3. Image 3 ie smallstrelitzia.jpg segmented into 10, 20 and 50 segments

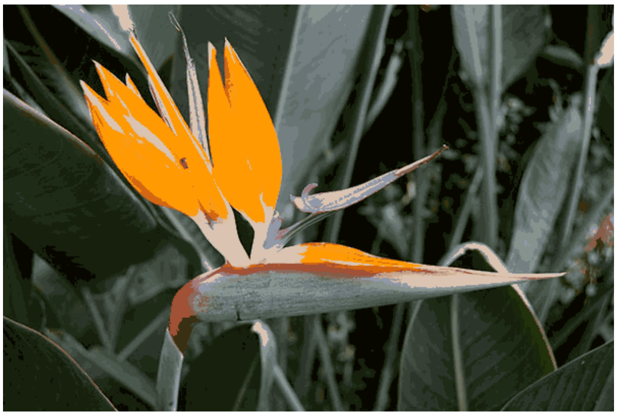
(1) **10**



(2) **20**

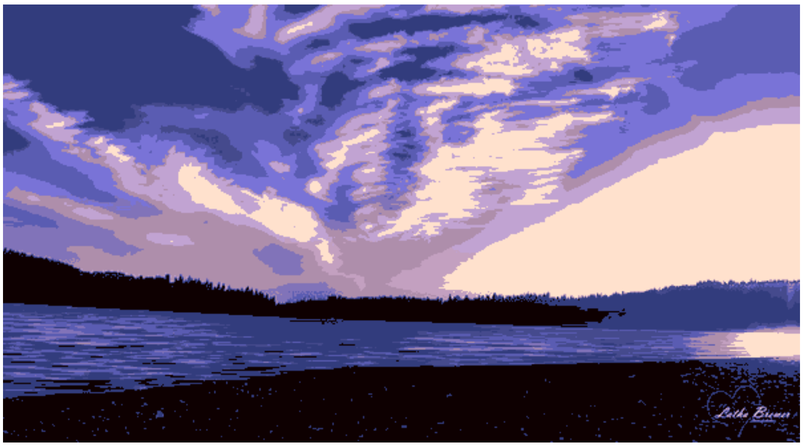


(3) **50**

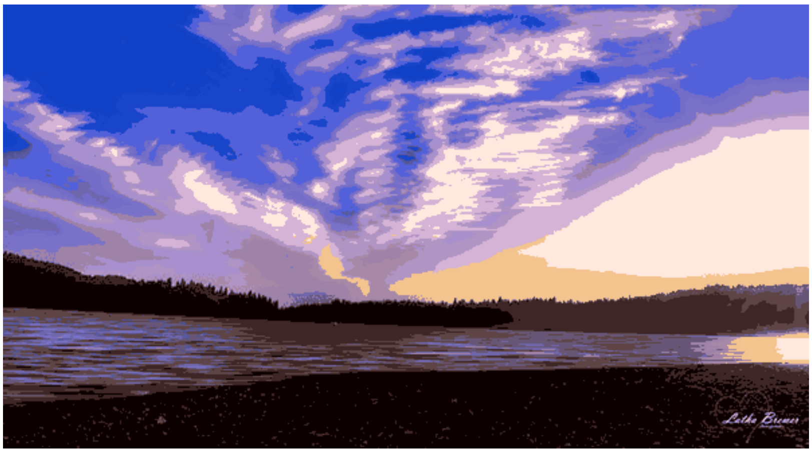


4. Image 4 ie smallsunset.jpg segmented into 10,20 and 50 segments

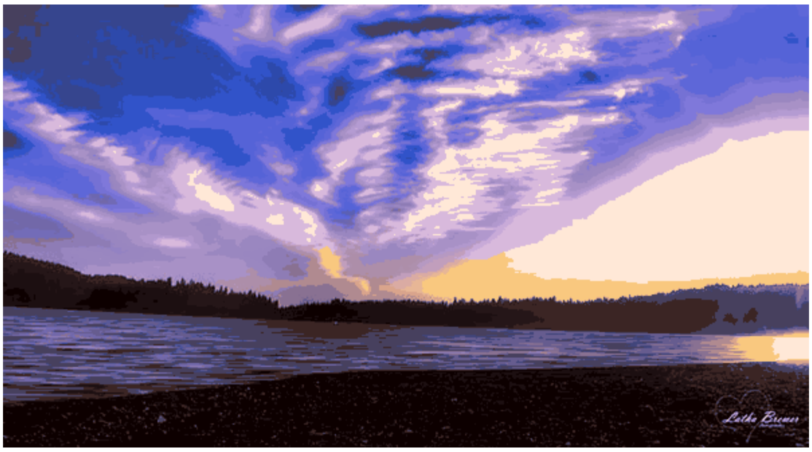
(4) **10**



(5) **20**

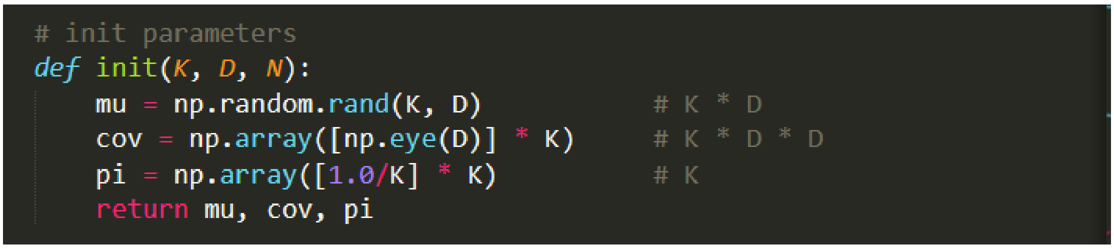


(6) **50**

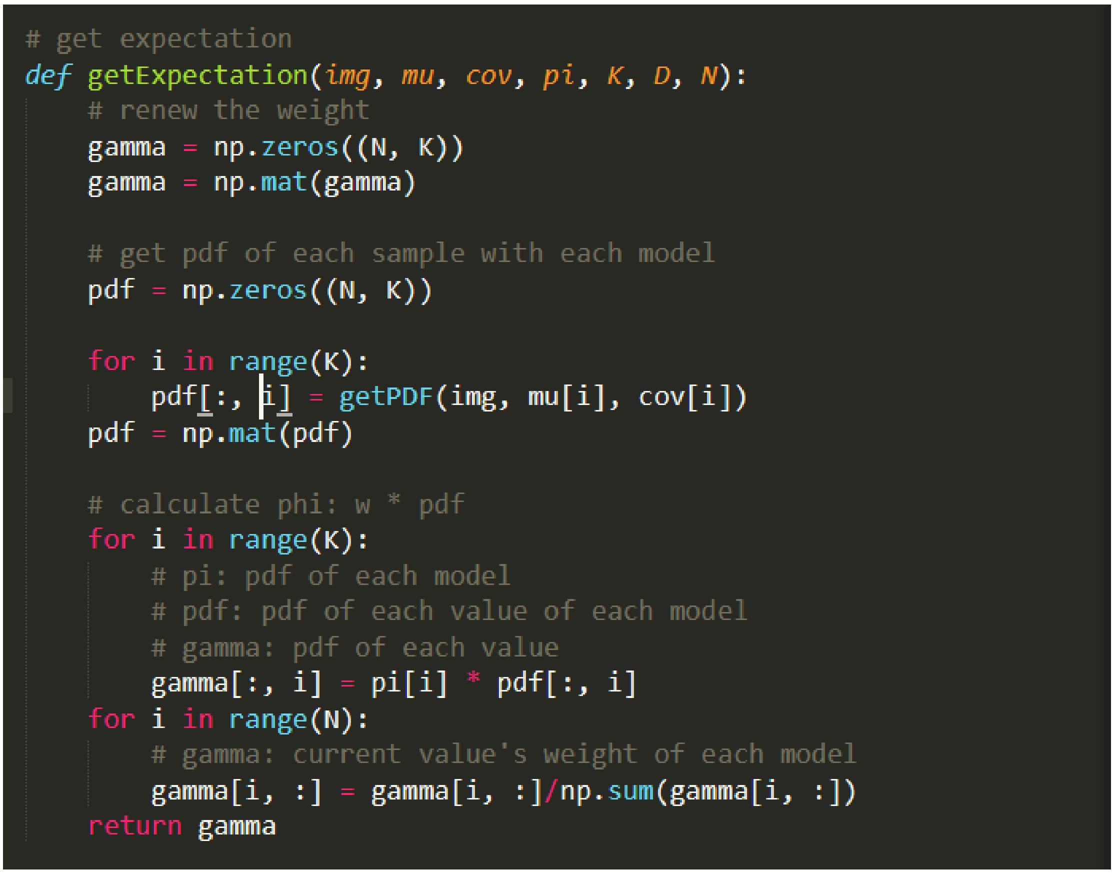


5. Display of tree image with 20 segments with 5 different initial points

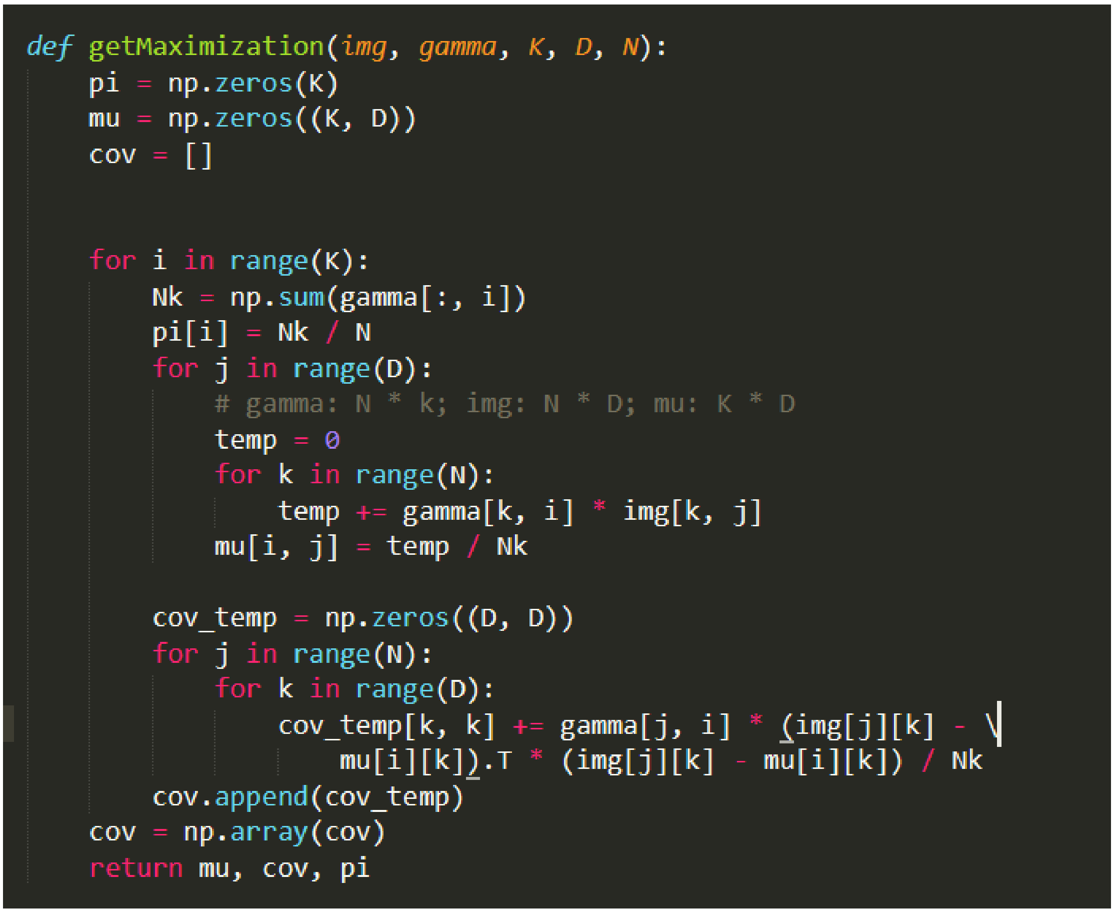
6. Code snippets for EM initialisation, updates



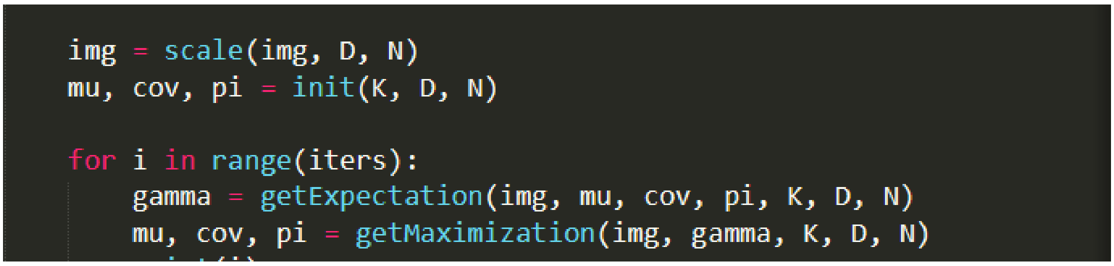
```
# init parameters
def init(K, D, N):
    mu = np.random.rand(K, D)           # K * D
    cov = np.array([np.eye(D)] * K)     # K * D * D
    pi = np.array([1.0/K] * K)          # K
    return mu, cov, pi
```



```
# get expectation
def getExpectation(img, mu, cov, pi, K, D, N):
    # renew the weight
    gamma = np.zeros((N, K))
    gamma = np.mat(gamma)

    # get pdf of each sample with each model
    pdf = np.zeros((N, K))

    for i in range(K):
        pdf[:, i] = getPDF(img, mu[i], cov[i])
    pdf = np.mat(pdf)

    # calculate phi: w * pdf
    for i in range(K):
        # pi: pdf of each model
        # pdf: pdf of each value of each model
        # gamma: pdf of each value
        gamma[:, i] = pi[i] * pdf[:, i]
    for i in range(N):
        # gamma: current value's weight of each model
        gamma[i, :] = gamma[i, :]/np.sum(gamma[i, :])
    return gamma
```

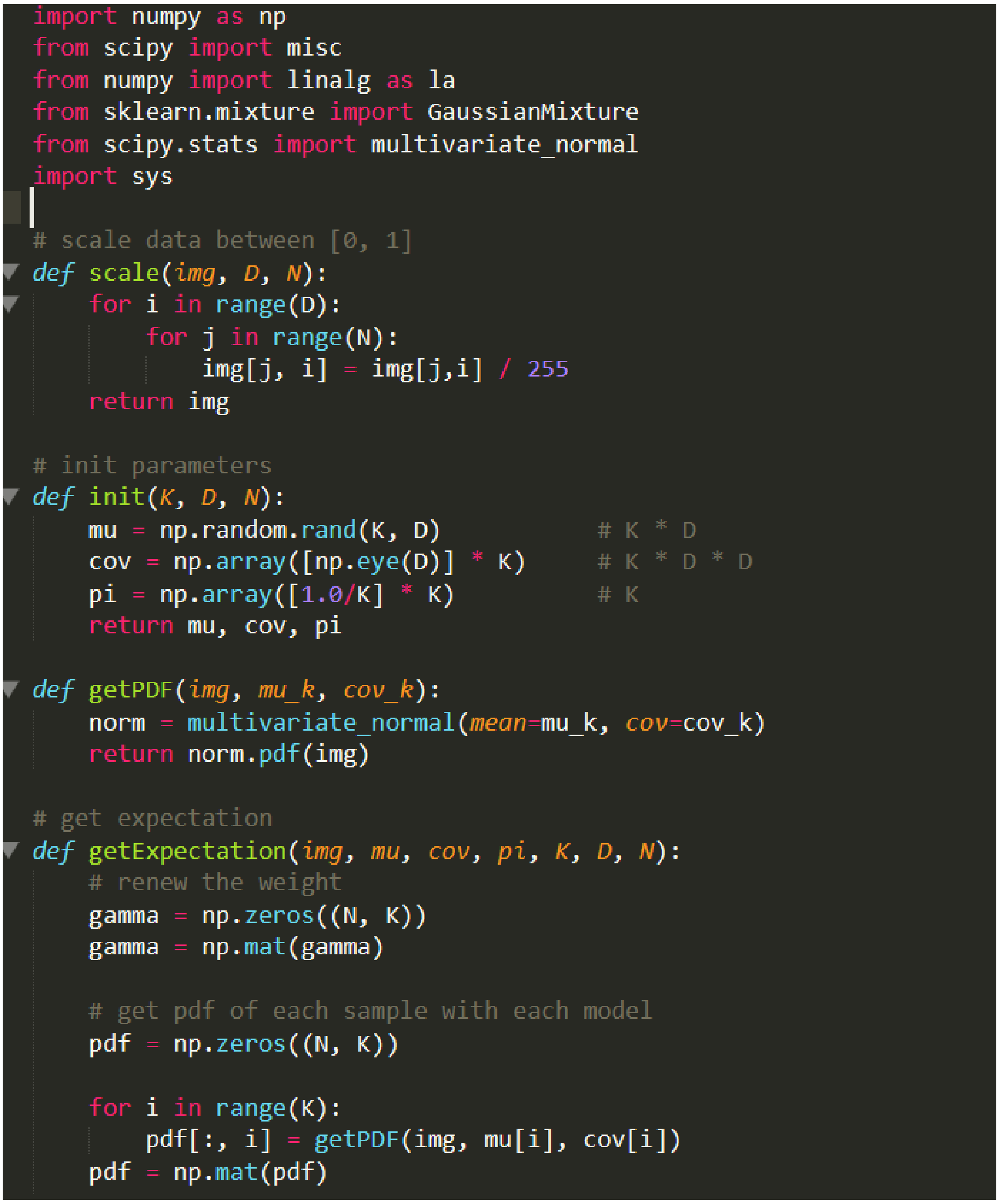


```python
def getMaximization(img, gamma, K, D, N):
    pi = np.zeros(K)
    mu = np.zeros((K, D))
    cov = []


    for i in range(K):
        Nk = np.sum(gamma[:, i])
        pi[i] = Nk / N
        for j in range(D):
            # gamma: N * k; img: N * D; mu: K * D
            temp = 0
            for k in range(N):
                temp += gamma[k, i] * img[k, j]
            mu[i, j] = temp / Nk

        cov_temp = np.zeros((D, D))
        for j in range(N):
            for k in range(D):
                cov_temp[k, k] += gamma[j, i] * (img[j][k] - \
                    mu[i][k]).T * (img[j][k] - mu[i][k]) / Nk
        cov.append(cov_temp)
    cov = np.array(cov)
    return mu, cov, pi
```

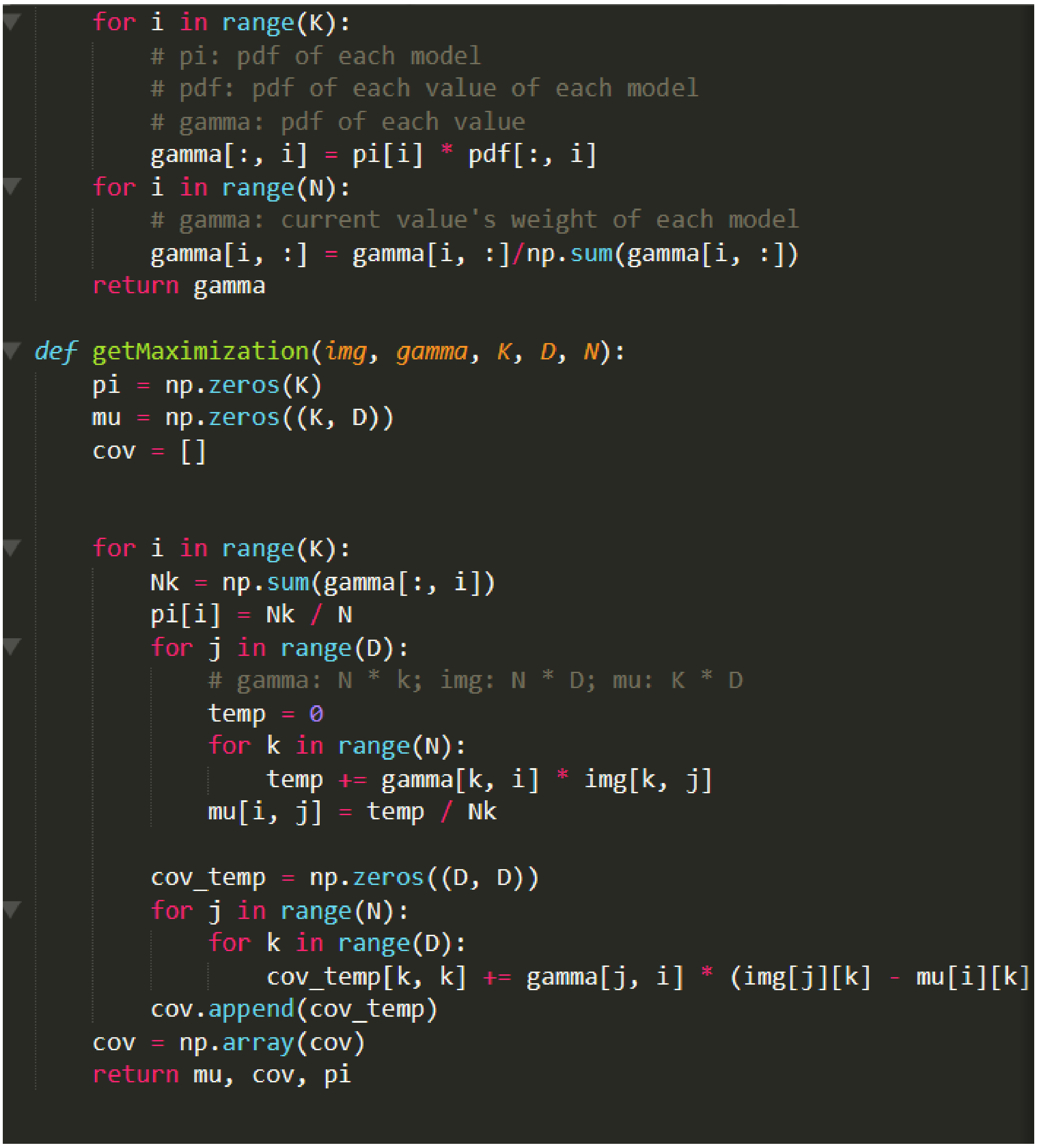


```python
    img = scale(img, D, N)
    mu, cov, pi = init(K, D, N)

    for i in range(iters):
        gamma = getExpectation(img, mu, cov, pi, K, D, N)
        mu, cov, pi = getMaximization(img, gamma, K, D, N)
```

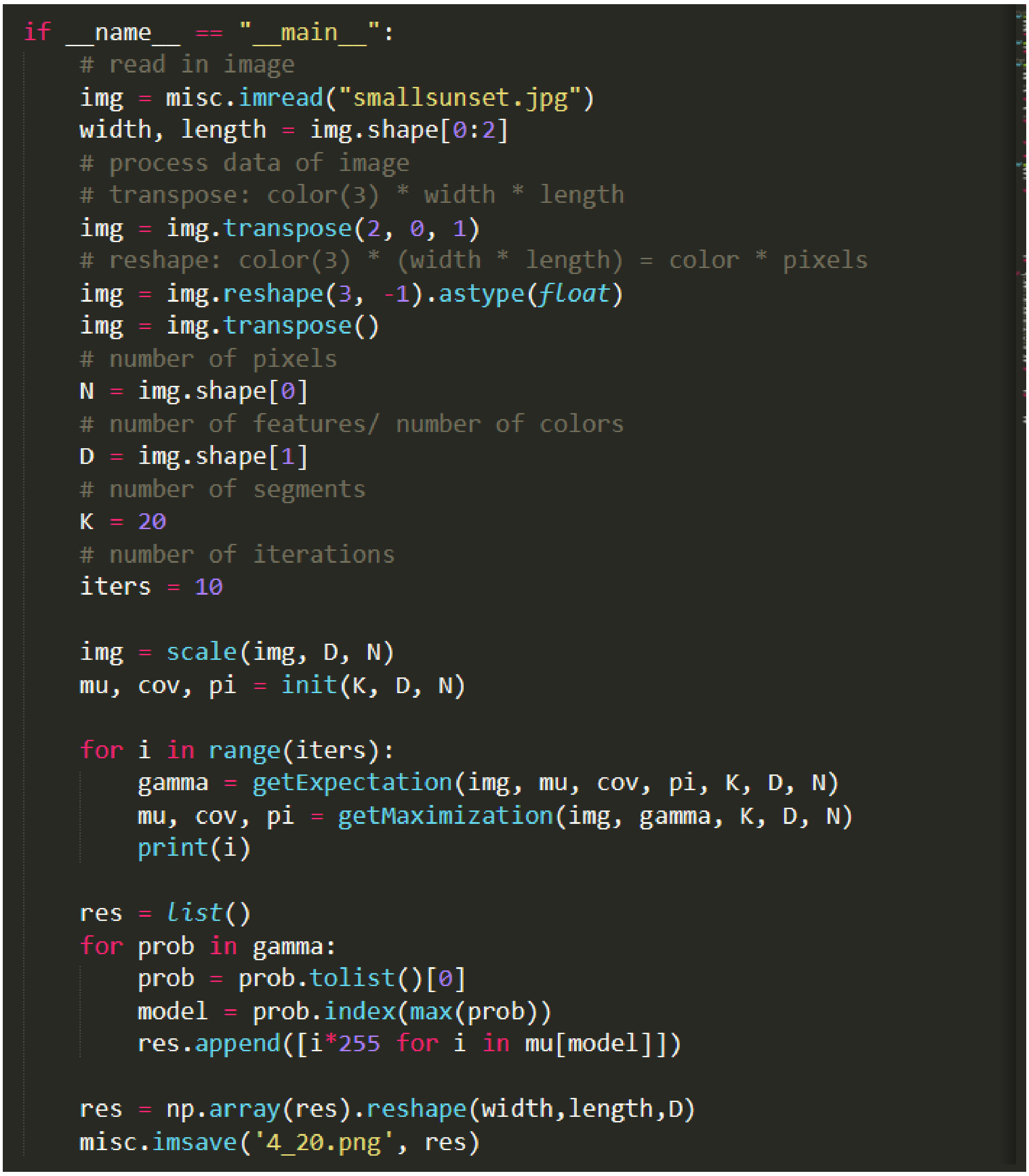
7. Other relevant code (optional)



```python
import numpy as np
from scipy import misc
from numpy import linalg as la
from sklearn.mixture import GaussianMixture
from scipy.stats import multivariate_normal
import sys

# scale data between [0, 1]
def scale(img, D, N):
    for i in range(D):
        for j in range(N):
            img[j, i] = img[j,i] / 255
    return img

# init parameters
def init(K, D, N):
    mu = np.random.rand(K, D)             # K * D
    cov = np.array([np.eye(D)] * K)       # K * D * D
    pi = np.array([1.0/K] * K)            # K
    return mu, cov, pi

def getPDF(img, mu_k, cov_k):
    norm = multivariate_normal(mean=mu_k, cov=cov_k)
    return norm.pdf(img)

# get expectation
def getExpectation(img, mu, cov, pi, K, D, N):
    # renew the weight
    gamma = np.zeros((N, K))
    gamma = np.mat(gamma)

    # get pdf of each sample with each model
    pdf = np.zeros((N, K))

    for i in range(K):
        pdf[:, i] = getPDF(img, mu[i], cov[i])
    pdf = np.mat(pdf)
```



```
    for i in range(K):
        # pi: pdf of each model
        # pdf: pdf of each value of each model
        # gamma: pdf of each value
        gamma[:, i] = pi[i] * pdf[:, i]
    for i in range(N):
        # gamma: current value's weight of each model
        gamma[i, :] = gamma[i, :]/np.sum(gamma[i, :])
    return gamma

def getMaximization(img, gamma, K, D, N):
    pi = np.zeros(K)
    mu = np.zeros((K, D))
    cov = []


    for i in range(K):
        Nk = np.sum(gamma[:, i])
        pi[i] = Nk / N
        for j in range(D):
            # gamma: N * k; img: N * D; mu: K * D
            temp = 0
            for k in range(N):
                temp += gamma[k, i] * img[k, j]
            mu[i, j] = temp / Nk

        cov_temp = np.zeros((D, D))
        for j in range(N):
            for k in range(D):
                cov_temp[k, k] += gamma[j, i] * (img[j][k] - mu[i][k]
        cov.append(cov_temp)
    cov = np.array(cov)
    return mu, cov, pi
```



```python
if __name__ == "__main__":
    # read in image
    img = misc.imread("smallsunset.jpg")
    width, length = img.shape[0:2]
    # process data of image
    # transpose: color(3) * width * length
    img = img.transpose(2, 0, 1)
    # reshape: color(3) * (width * length) = color * pixels
    img = img.reshape(3, -1).astype(float)
    img = img.transpose()
    # number of pixels
    N = img.shape[0]
    # number of features/ number of colors
    D = img.shape[1]
    # number of segments
    K = 20
    # number of iterations
    iters = 10

    img = scale(img, D, N)
    mu, cov, pi = init(K, D, N)

    for i in range(iters):
        gamma = getExpectation(img, mu, cov, pi, K, D, N)
        mu, cov, pi = getMaximization(img, gamma, K, D, N)
        print(i)

    res = list()
    for prob in gamma:
        prob = prob.tolist()[0]
        model = prob.index(max(prob))
        res.append([i*255 for i in mu[model]])

    res = np.array(res).reshape(width,length,D)
    misc.imsave('4_20.png', res)
```