**Contents**

Question 1 -------------------------------------------------------------------------------------------- 2

1. Image of code --------------------------------------------------------------------------------- 2

2. Code ---------------------------------------------------------------------------------------------- 3

3. Test ----------------------------------------------------------------------------------------------- 4

Question 2 -------------------------------------------------------------------------------------------- 4

1. Image of code --------------------------------------------------------------------------------- 4

2. Code ----------------------------------------------------------------------------------------------5

3. Test ----------------------------------------------------------------------------------------------- 7

Question 3 -------------------------------------------------------------------------------------------- 7

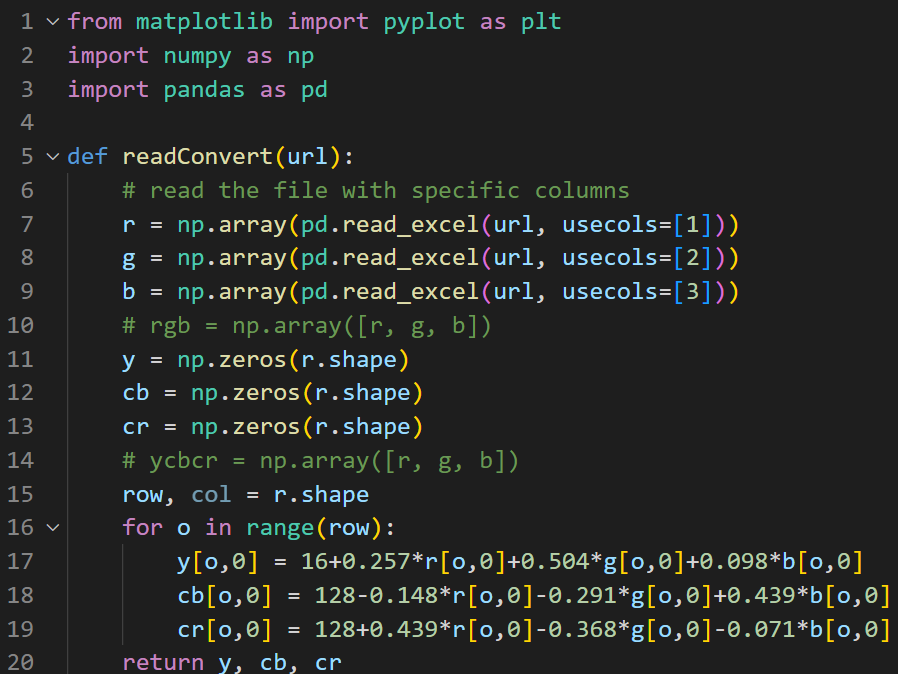
1. Image of code --------------------------------------------------------------------------------- 7

2. Code ----------------------------------------------------------------------------------------------9

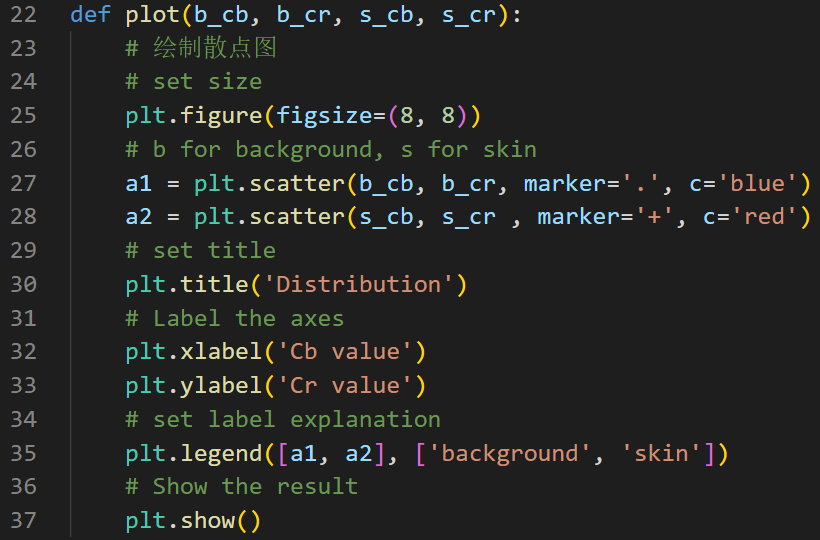
3. Test ----------------------------------------------------------------------------------------------- 11

**Question 1**

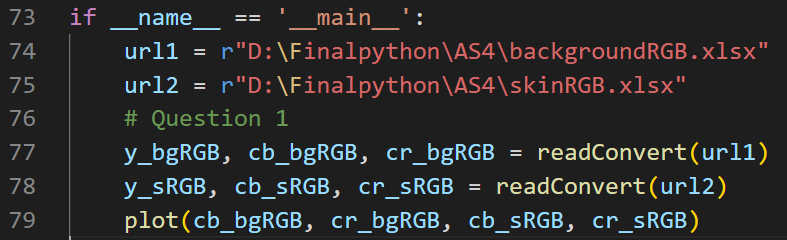
1. **Image of code for the better reading experience**
2. Function to convert RGB value to YCbCr value



1. Function to plot the distribution based on Cb and Cr value



1. Main function used to test the two functions above



1. **Code details**

from matplotlib import pyplot as plt

import numpy as np

import pandas as pd

def readConvert(url):

# read the file with specific columns

r = np.array(pd.read\_excel(url, usecols=[1]))

g = np.array(pd.read\_excel(url, usecols=[2]))

b = np.array(pd.read\_excel(url, usecols=[3]))

# rgb = np.array([r, g, b])

y = np.zeros(r.shape)

cb = np.zeros(r.shape)

cr = np.zeros(r.shape)

# ycbcr = np.array([r, g, b])

row, col = r.shape

for o in range(row):

y[o,0] = 16+0.257\*r[o,0]+0.504\*g[o,0]+0.098\*b[o,0]

cb[o,0] = 128-0.148\*r[o,0]-0.291\*g[o,0]+0.439\*b[o,0]

cr[o,0] = 128+0.439\*r[o,0]-0.368\*g[o,0]-0.071\*b[o,0]

return y, cb, cr

def plot(b\_cb, b\_cr, s\_cb, s\_cr):

# 绘制散点图

# set size

plt.figure(figsize=(8, 8))

# b for background, s for skin

a1 = plt.scatter(b\_cb, b\_cr, marker='.', c='blue')

a2 = plt.scatter(s\_cb, s\_cr , marker='+', c='red')

# set title

plt.title('Distribution')

# Label the axes

plt.xlabel('Cb value')

plt.ylabel('Cr value')

# set label explanation

plt.legend([a1, a2], ['background', 'skin'])

# Show the result

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

url1 = r"D:\Finalpython\AS4\backgroundRGB.xlsx"

url2 = r"D:\Finalpython\AS4\skinRGB.xlsx"

# Question 1

y\_bgRGB, cb\_bgRGB, cr\_bgRGB = readConvert(url1)

y\_sRGB, cb\_sRGB, cr\_sRGB = readConvert(url2)

plot(cb\_bgRGB, cr\_bgRGB, cb\_sRGB, cr\_sRGB)

1. **Test**

The figures below is the result when using main function to test

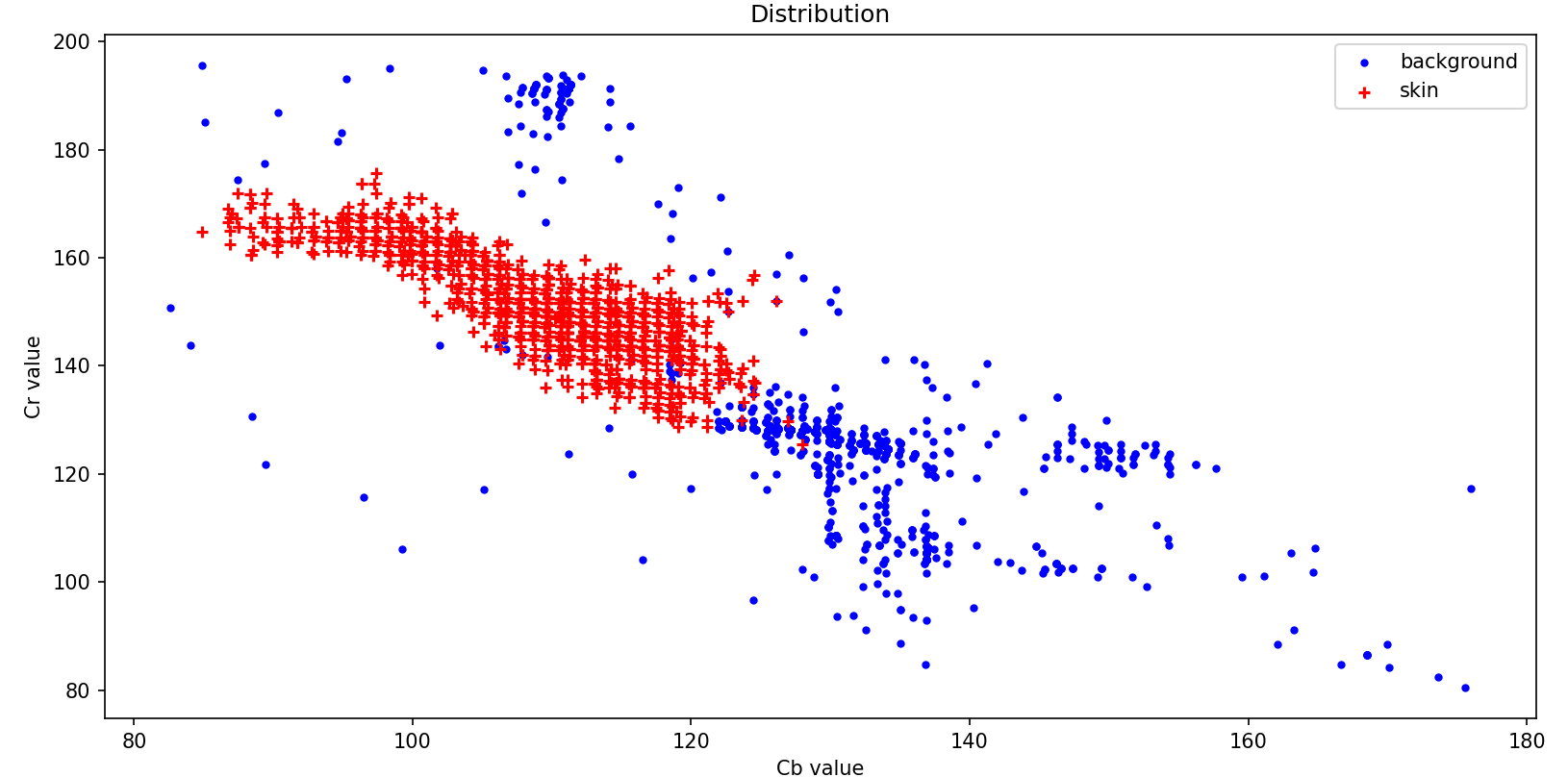
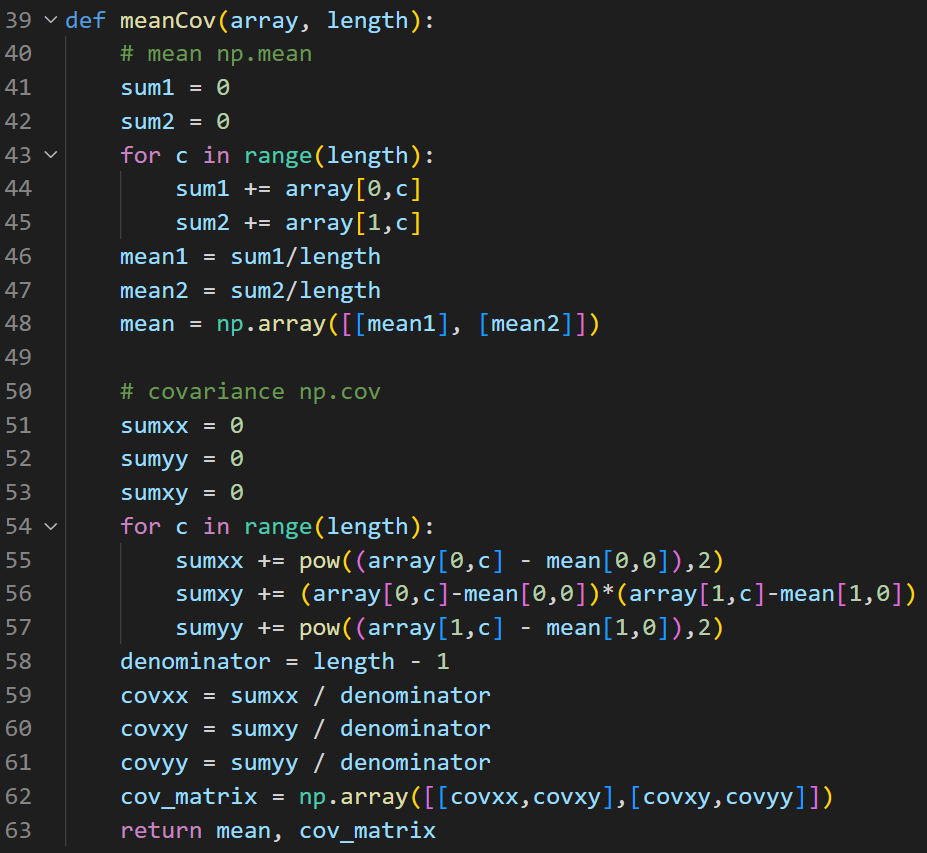


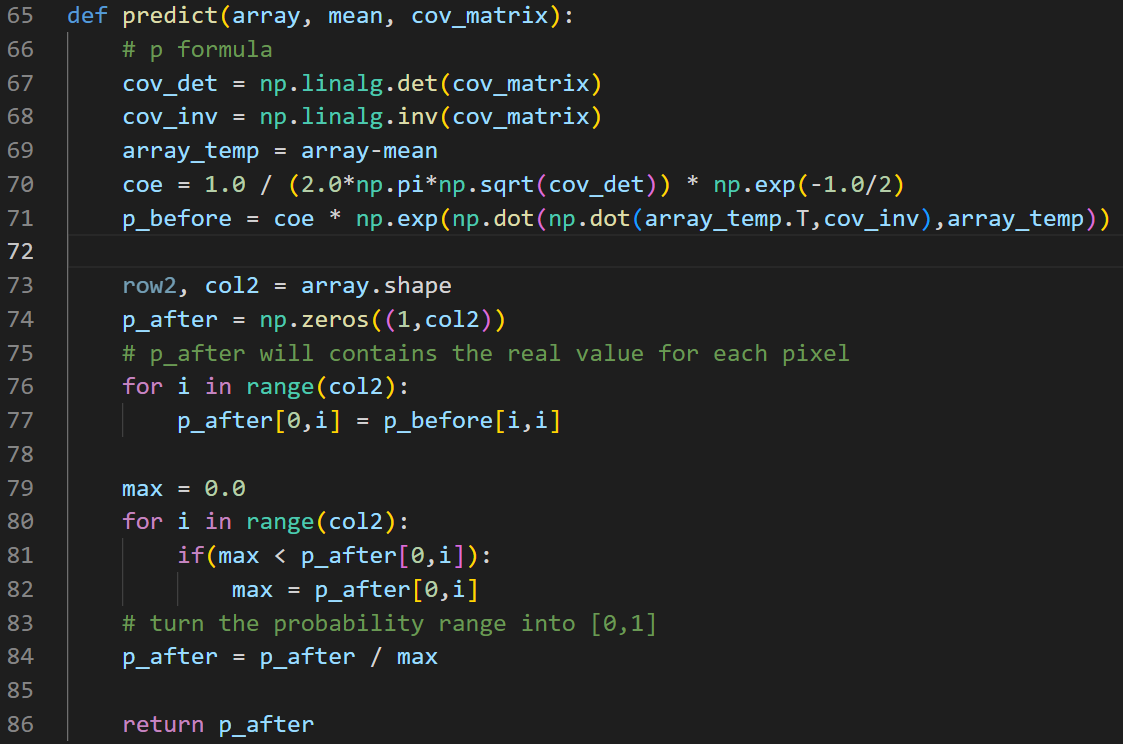
Figure 1: Distribution of skin color based on Cb and Cr values

**Question 2**

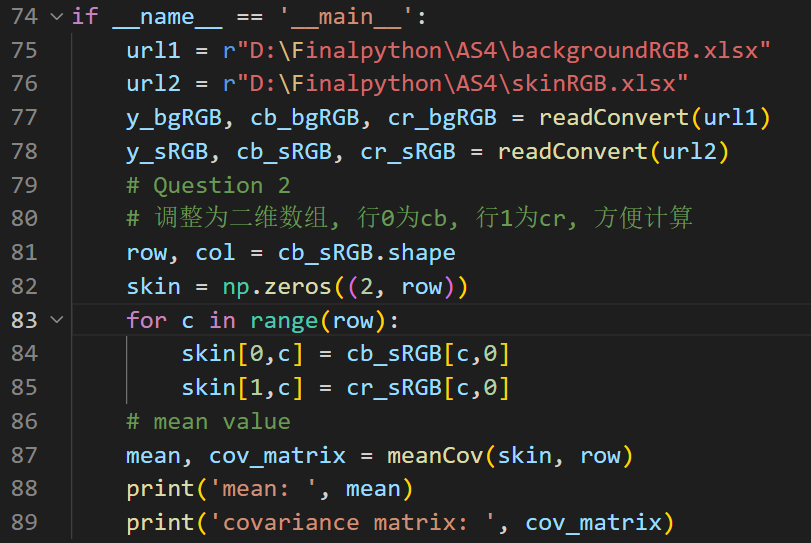
1. **Image of code for the better reading experience**
2. Function to calculate the mean and covariance matrix



1. Function to implement Gaussian Mixture Model



1. Complete main function to test the function above



1. **Code details**

def meanCov(array, length):

# mean np.mean

sum1 = 0

sum2 = 0

for c in range(length):

sum1 += array[0,c]

sum2 += array[1,c]

mean1 = sum1/length

mean2 = sum2/length

mean = np.array([[mean1], [mean2]])

# covariance np.cov

sumxx = 0

sumyy = 0

sumxy = 0

for c in range(length):

sumxx += pow((array[0,c] - mean[0,0]),2)

sumxy += (array[0,c]-mean[0,0])\*(array[1,c]-mean[1,0])

sumyy += pow((array[1,c] - mean[1,0]),2)

denominator = length - 1

covxx = sumxx / denominator

covxy = sumxy / denominator

covyy = sumyy / denominator

cov\_matrix = np.array([[covxx,covxy],[covxy,covyy]])

return mean, cov\_matrix

def predict(array, mean, cov\_matrix):

# p formula

cov\_det = np.linalg.det(cov\_matrix)

cov\_inv = np.linalg.inv(cov\_matrix)

array\_temp = array-mean

coe = 1.0 / (2.0\*np.pi\*np.sqrt(cov\_det)) \* np.exp(-1.0/2)

p\_before=coe\*np.exp(np.dot(np.dot(array\_temp.T,cov\_inv),array\_temp))

row2, col2 = array.shape

p\_after = np.zeros((1,col2))

# p\_after will contains the real value for each pixel

for i in range(col2):

p\_after[0,i] = p\_before[i,i]

max = 0.0

for i in range(col2):

if(max < p\_after[0,i]):

max = p\_after[0,i]

# turn the probability range into [0,1]

p\_after = p\_after / max

return p\_after

if \_\_name\_\_ == '\_\_main\_\_':

url1 = r"D:\Finalpython\AS4\backgroundRGB.xlsx"

url2 = r"D:\Finalpython\AS4\skinRGB.xlsx"

# Question 1

y\_bgRGB, cb\_bgRGB, cr\_bgRGB = readConvert(url1)

y\_sRGB, cb\_sRGB, cr\_sRGB = readConvert(url2)

y\_bRGB, cb\_bRGB, cr\_bRGB = readConvert(url2)

row1, col1 = cb\_sRGB.shape

skin = np.zeros((2, row1))

for c1 in range(row1):

skin[0,c1] = cb\_sRGB[c1,0]

skin[1,c1] = cr\_sRGB[c1,0]

mean1, cov\_matrix1 = meanCov(skin, row1)

print("mean", mean1)

print("Covariance matrix", cov\_matrix1)

1. **Test**

The result tests the function “meanCov” based on “skinRGB.xlsx”,

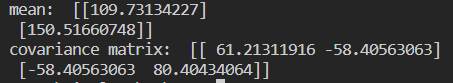


Figure 2: Mean and covariance matrix results based on skin

*Mention: Since the question 3 will show two probability maps, it is convenient if we calculate the mean and covariance matrix based on backgroundRGB.xlsx. Hence, although question2 only asked to calculate based on skin, the calculation based on background is also shown below.*

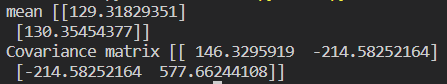
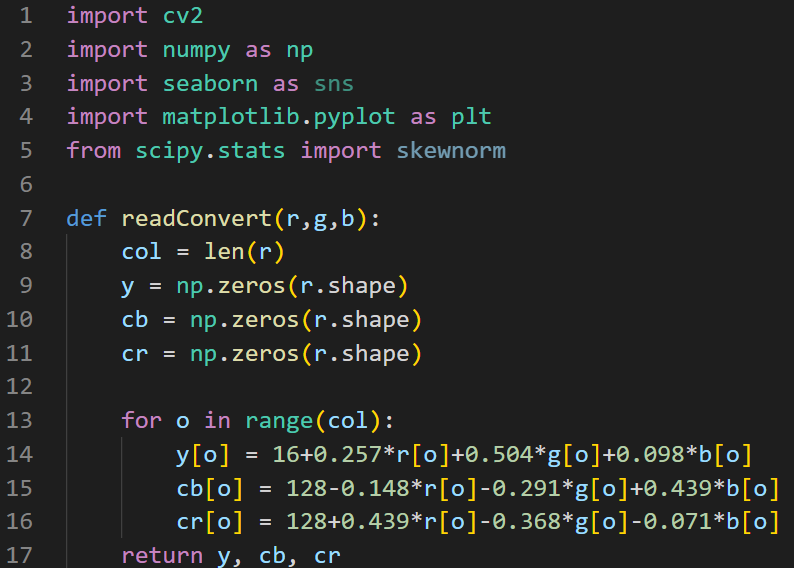


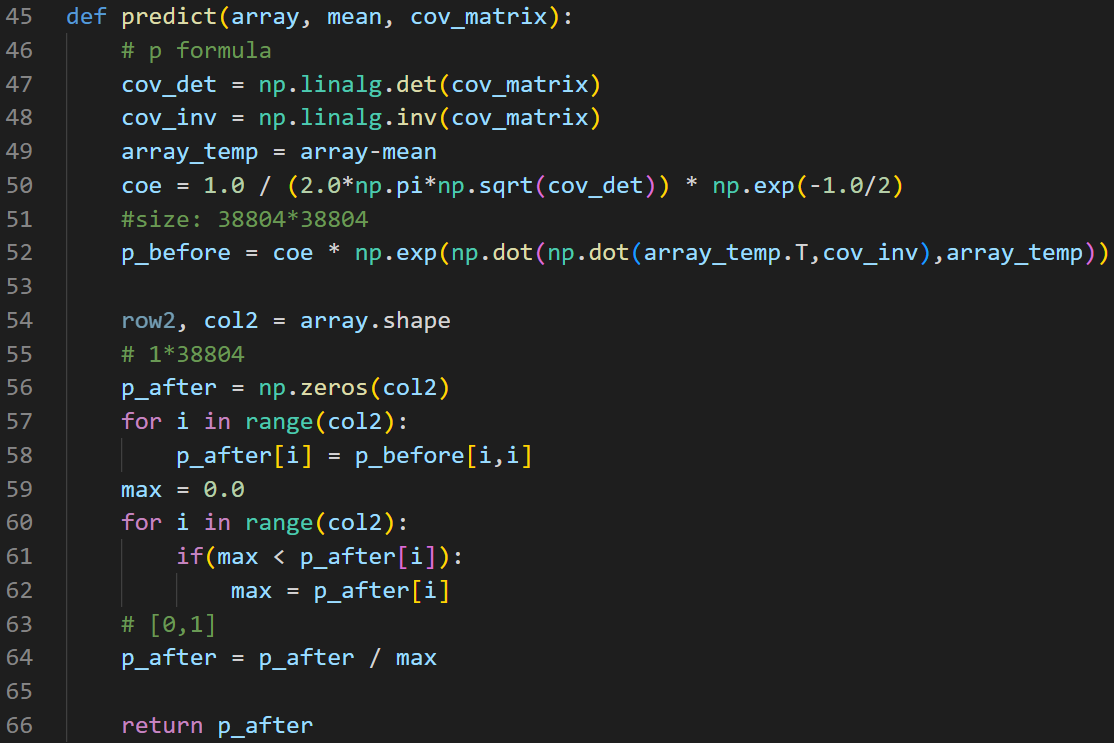
Figure 2: Mean and covariance matrix results based on background

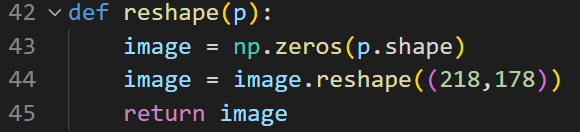
**Question 3**

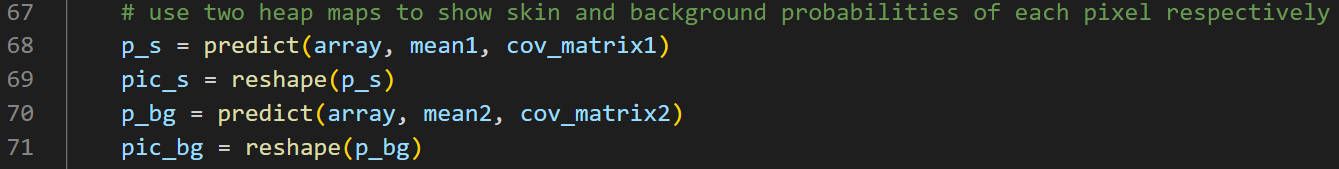
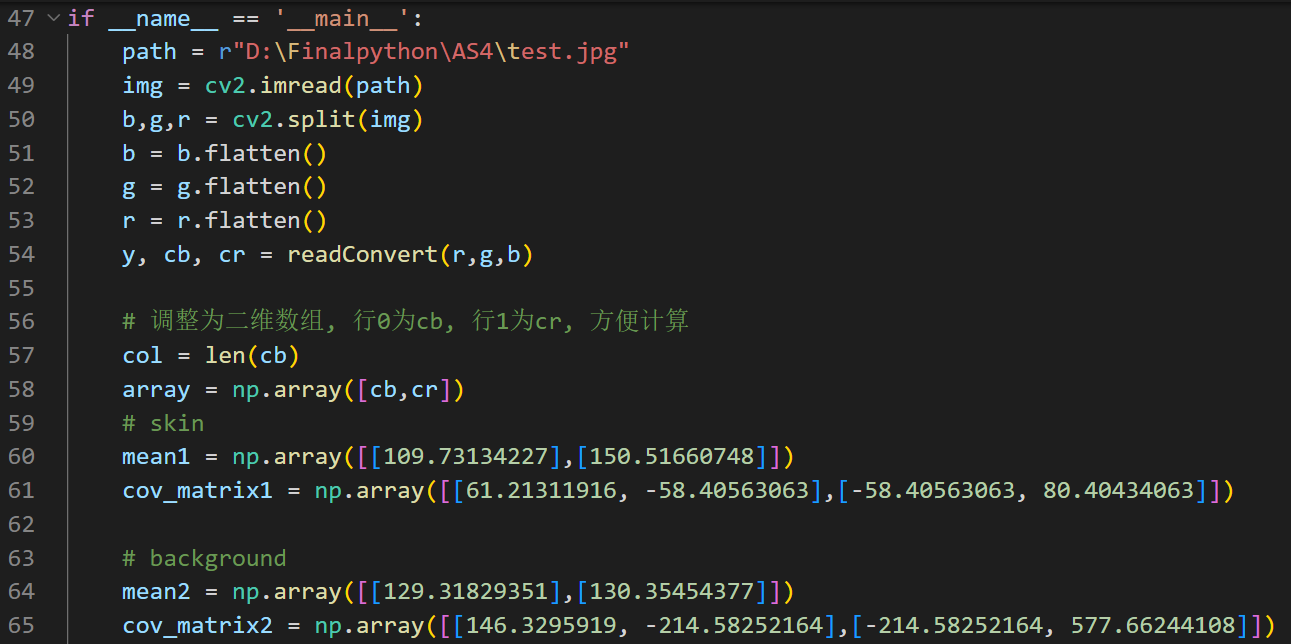
1. **Image of code for the better reading experience**

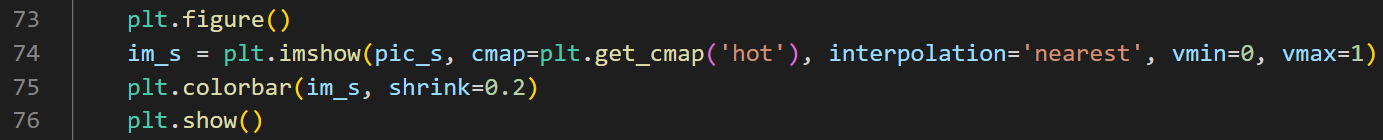
The following is the complete code for Question 3 since some functions are modifies to fit the situation (using image).











1. **Code details**

import cv2

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import skewnorm

def readConvert(r,g,b):

col = len(r)

y = np.zeros(r.shape)

cb = np.zeros(r.shape)

cr = np.zeros(r.shape)

for o in range(col):

y[o] = 16+0.257\*r[o]+0.504\*g[o]+0.098\*b[o]

cb[o] = 128-0.148\*r[o]-0.291\*g[o]+0.439\*b[o]

cr[o] = 128+0.439\*r[o]-0.368\*g[o]-0.071\*b[o]

return y, cb, cr

def predict(array, mean, cov\_matrix):

# p formula

cov\_det = np.linalg.det(cov\_matrix)

cov\_inv = np.linalg.inv(cov\_matrix)

array\_temp = array-mean

coe = 1.0 / (2.0\*np.pi\*np.sqrt(cov\_det)) \* np.exp(-1.0/2)

#size: 38804\*38804

p\_before = coe \* np.exp(np.dot(np.dot(array\_temp.T,cov\_inv),array\_temp))

row2, col2 = array.shape

# 1\*38804

p\_after = np.zeros(col2)

for i in range(col2):

p\_after[i] = p\_before[i,i]

max = 0.0

for i in range(col2):

if(max < p\_after[i]):

max = p\_after[i]

# [0,1]

p\_after = p\_after / max

return p\_after

def reshape(p):

image = np.zeros(p.shape)

image = image.reshape((218,178))

return image

if \_\_name\_\_ == '\_\_main\_\_':

path = r"D:\Finalpython\AS4\test.jpg"

img = cv2.imread(path)

b,g,r = cv2.split(img)

b = b.flatten()

g = g.flatten()

r = r.flatten()

y, cb, cr = readConvert(r,g,b)

# 调整为二维数组, 行0为cb, 行1为cr, 方便计算

col = len(cb)

array = np.array([cb,cr])

# skin

mean1 = np.array([[109.73134227],[150.51660748]])

cov\_matrix1 = np.array([[61.21311916, -58.40563063],[-58.40563063, 80.40434063]])

# background

mean2 = np.array([[129.31829351],[130.35454377]])

cov\_matrix2 = np.array([[146.3295919, -214.58252164],[-214.58252164, 577.66244108]])

# use two heap maps to show skin and background probabilities of each pixel respectively

p\_s = predict(array, mean1, cov\_matrix1)

pic\_s = reshape(p\_s)

p\_bg = predict(array, mean2, cov\_matrix2)

pic\_bg = reshape(p\_bg)

plt.figure()

im\_s = plt.imshow(pic\_s, cmap=plt.get\_cmap('hot'), interpolation='nearest', vmin=0, vmax=1)

plt.colorbar(im\_s, shrink=0.2)

plt.show()

plt.figure()

im\_bg = plt.imshow(pic\_bg, cmap=plt.get\_cmap('hot'), interpolation='nearest', vmin=0, vmax=1)

plt.colorbar(im\_bg, shrink=0.2)

plt.show()

img\_gray = pic\_s \* 255

cv2.imshow('gray', img\_gray)

cv2.waitKey(0)

cv2.destroyAllWindows()

1. **Test**

*The gray image is shown based on the skin probability.*

*Using the means and covariance matrixes got from question 2, the skin and background probabilities of each pixel are calculated, which lead to the heap map with the same size (218\*178) of the original image.*

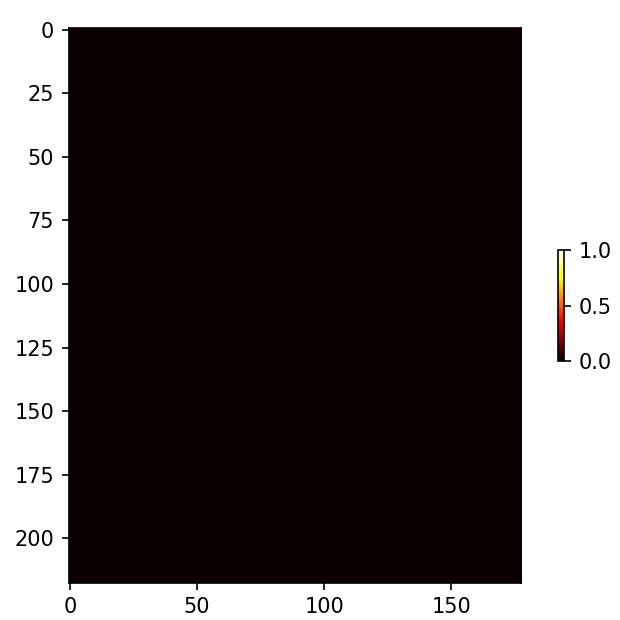
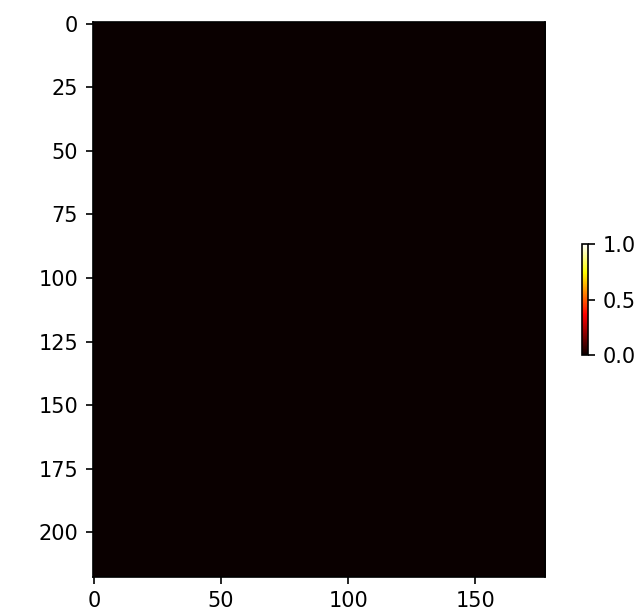


Figure 3: probability maps of skin (left) and background (right)



Figure 4: gray picture based on skin probability.