Exercise 3.3

Image Segmentation via Expectation Maximization

In this task, we apply EM algorithm to segment images. The intuition is to assign every pixels of the image to a class/cluster based on their colors (color image has 3 channels). The class/cluster can be thought as latent variable of the pixels, in which the pixels have similar colors.

Necessary imports

In [1]:

```
import numpy as np
import scipy.stats
import math
import cv2
from scipy import ndimage
# Try Kmeans from diffrerent packages
from sklearn import cluster
from scipy.cluster.vq import kmeans2
import matplotlib.pyplot as plt
from scipy import ndimage
%matplotlib inline

sample_image = 'flower.jpg'
```

Image processings

In [2]:

```
# input 3d array >> output 2d array
def flatten_img(img_3d):
  '''
  Flatten 3-dim image [w, h, c] into 2-dim array [w x h, c]

  Args:
    img_3d: Image array [w, h, c]

  Returns:
    img_2d: flatten image [w x h, c]
  '''
  x, y, z = img_3d.shape
  img_2d = img_3d.reshape(x*y, z)
  img_2d = np.array(img_2d, dtype=float)
  return img_2d

def recover_img(img_2d, w, h, c):
  '''
  Recover 2-dim image [w x h, c] into original 3-dim array [w, h, c]

  Args:
    img_2d: [w, h, c]
    w, h, c: width, height and channels of image to recover

  Returns:
    img_3d: recovered image [w, h, c]
  '''
  recover_img = img_2d.reshape(w, h, c)
  return recover_img
```

EM implementation with KMeans Clustering Initialization

Exercise 3.3.1

Implement the EM algorithm with KMeans Clustering initialization. Feel free to reuse your EM implementation from previous exercise here.

In [3]:
```
# EM Algorithm
# TODO: Your code here
def expectation(data, means, covs, pis):
    n, d = data.shape
    k = len(covs)
    resps = np.zeros((k,n))
    for j in range(k):
        resps[j] = pis[j] * scipy.stats.multivariate_normal.pdf(data, mean=means
    resps = resps / np.sum(resps, axis=0)
    return resps

def maximization(data, resps):
    n, d = data.shape
    k = resps.shape[0]
    n_k = np.sum(resps, axis=1)
    new_means = resps @ data
    new_means = new_means / n_k[:,np.newaxis]
    new_covs = np.zeros((k,d,d))
    for j in range(k):
        diff = data - new_means[j]
        resp = resps[j]
        new_covs[j] = (resp * diff.T) @ diff
    new_covs = new_covs / n_k[:,np.newaxis,np.newaxis]
    new_pis = n_k / n
    return new_means, new_covs, new_pis

def evaluation(data, means, covs, pis):
    k = len(means)
    res = 0
    for j in range(k):
        row = pis[j] * scipy.stats.multivariate_normal.pdf(data, mean=means[j],
        res += row
    res = np.sum(np.log(res))
    return res

def em(data, means, covs, pis, eps):
    diff = np.inf
    ol = 0
    m = means
    c = covs
```

```python
    p = pis
    while diff > eps:
        resps = expectation(data, m, c, p)
        nm, nc, npi = maximization(data, resps)
        l = evaluation(data, nm, nc, npi)
        diff = np.abs(l-ol)
        ol = l
        m = nm
        c = nc
        p = npi
        print(diff,end='\r')
    print()
    print("EM finished.")
    return ol,p,m,c,resps

def init_covariance(data, centroids, labels):
    k, d = centroids.shape
    assert d == data.shape[1]
    new_covariances = np.zeros((k,d,d))
    ident_mat = np.identity(d)
    for i in range(k):
        centroid = centroids[i]
        cluster = np.where(labels == i)
        mean_dist = np.mean(np.linalg.norm(data[cluster] - centroid, axis=1))
        new_covariances[i] = mean_dist * ident_mat
    return new_covariances

def init_mix_coeffs(k):
    return 1/k * np.ones(k)

def em_with_kmeans(data, k, eps):
    kmeans = cluster.KMeans(n_clusters=k, n_init='auto')
    labels = kmeans.fit_predict(data)
    means = kmeans.cluster_centers_
    covs = init_covariance(data, means, labels)
    pis = init_mix_coeffs(k)
    return em(data, means, covs, pis, eps)
```

Image Segmentation with EM

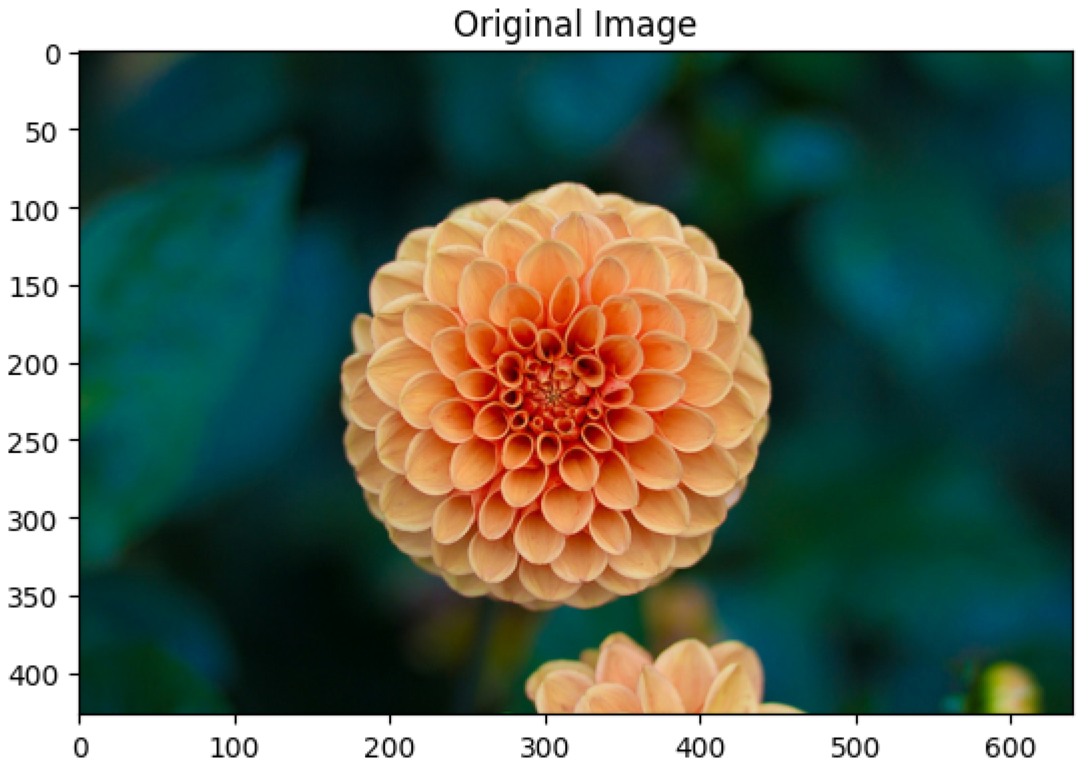
We experiment the implemented EM algorithm to segment the sample flower image from `sklearn`.

In [4]:

```python
from sklearn.datasets import load_sample_image

orig_img = load_sample_image(sample_image)
W, H, C = orig_img.shape
print('Size of sample image: ', orig_img.shape)
plt.imshow(orig_img)
plt.title('Original Image');
```

```
Size of sample image:  (427, 640, 3)
```

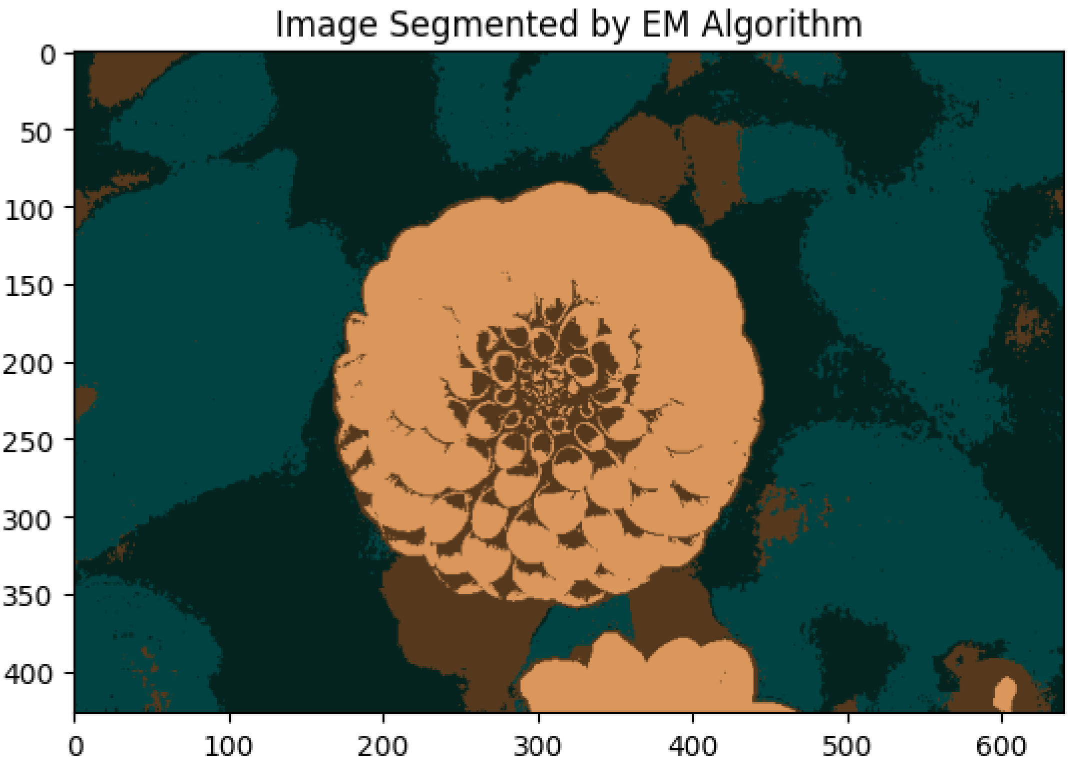


Exercise 3.3.2

Apply your EM implementation with four components (`k=4`) to segment the image and visualize the segments with the mean color of each segment.

Hint: You should flatten the image first and use the argmax operation on the responsibility vector to classify the segment of the pixels.

Try to match the following segmented image as closely as possible



In [5]:
```
# run EM on image for segmentation
# TODO: Your code here
np.random.seed(40)
img_2d = flatten_img(orig_img)
k = 4
eps = 0.00001
ol,p,m,c,resps = em_with_kmeans(img_2d, k, eps)
```

```
8.083414286375046e-065
EM finished.
```

In [6]:
```
# recover image with color segments using recover_img() above.
# Note that converting array type to int might be necessary
# TODO: Your code here
labels = np.argmax(resps, axis=0)
em_img = np.asarray(m[labels], dtype='int')
em_img = recover_img(em_img, W,H,C)
plt.imshow(em_img)
plt.title('Image Segmented by EM Algorithm');
```

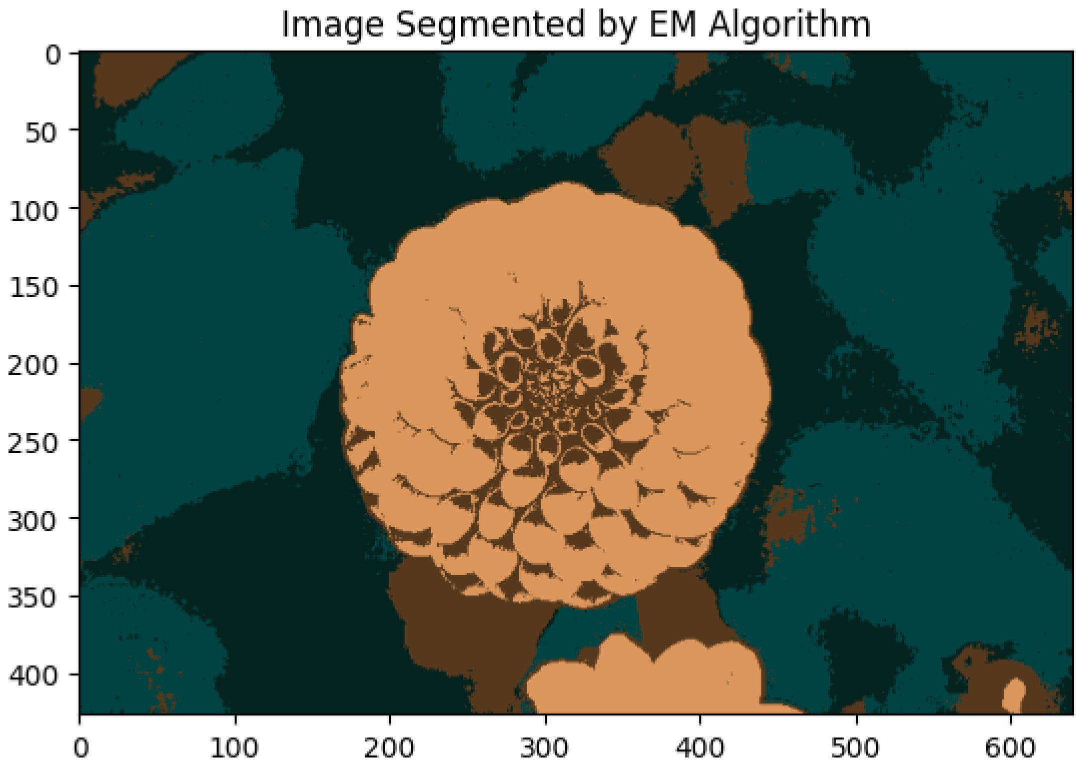


Image Segmented by EM Algorithm