In [244]:
```
# Wei Chen
import numpy as np
import matplotlib.pyplot as plt
```

In [245]:
```
# ===== Problem 2a =====
# Define Matrix M and Vectors a,b,c in Python with NumPy

M, a, b, c = None, None, None, None
```

In [246]:
```
M = np.array([[1,2,3],
              [4,5,6],
              [7,8,9],
              [0,2,2]])
a = np.array([[1],
              [1],
              [0]])


b = np.array([[-1],
              [2],
              [5]])
c = np.array([[0],
              [2],
              [3],
              [2]])
```

In [247]:
```
print(M)
print(a)
print(b)
print(c)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [0 2 2]]
[[1]
 [1]
 [0]]
[[-1]
 [ 2]
 [ 5]]
[[0]
 [2]
 [3]
 [2]]
```

In [248]:
```
# ===== Problem 2b =====
# Find the dot product of vectors a and b, save the value to aDotb

aDotb = None
```

In [249]:
```
aDotb = a.T.dot(b)
print(aDotb)
```

```
[[1]]
```

In [250]:
```
# ===== Problem 2c =====
# Find the element-wise product of a and b
```

In [251]:
```
print(np.multiply(a,b))
```

```
[[-1]
 [ 2]
 [ 0]]
```

In [252]:
```
# ===== Problem 2d =====
# Find (a^T b)Ma
```

In [253]:
```
a.T.dot(b)*M.dot(a)
```

Out[253]:
```
array([[ 3],
       [ 9],
       [15],
       [ 2]])
```

In [254]:
```
# ===== Problem 2e =====
# Without using a loop, multiply each row of M element-wise by a.
# Hint: The function repmat() may come in handy.

newM = None
```

In [255]:
```
M = np.multiply(M, np.tile(a.T,(4,1)))
print(M)
```

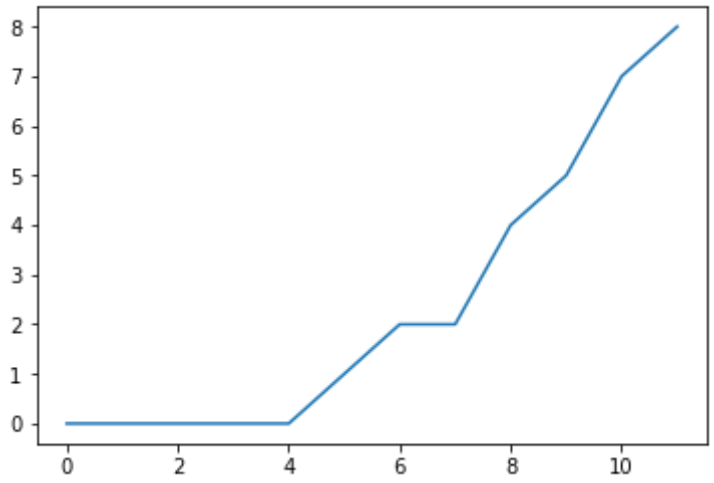
```
[[1 2 0]
 [4 5 0]
 [7 8 0]
 [0 2 0]]
```

In [256]:
```
# ===== Problem 2f =====
# Without using a loop, sort all of the values
# of M in increasing order and plot them.
# Note we want you to use newM from e.}
```

In [257]:
```
M = M.flatten()
M.sort()
print(M)
```

```
[0 0 0 0 0 1 2 2 4 5 7 8]
```

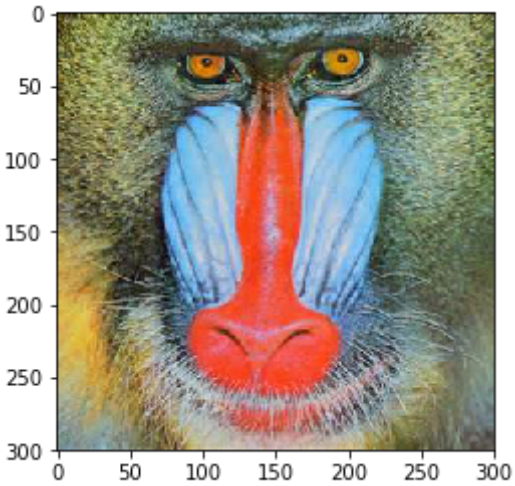
In [258]:
```
plt.plot(M)
plt.show()
```



In [259]:
```
#3a
# Read in the images, image1.jpg and image2.jpg, as color images.
from scipy import misc
```

In [260]:
```
img1 = misc.imread('image1.jpg', flatten=False)
img2 = misc.imread('image2.jpg', flatten=False)
```

In [261]:
```
plt.imshow(img1)
plt.show()
```



In [262]:
```
plt.imshow(img2)
plt.show()
```

In [263]:
```
#3b
#Add the images together and re-normalize them to have minimum value 0
# and maximum value 1. Display this image in your report.
```

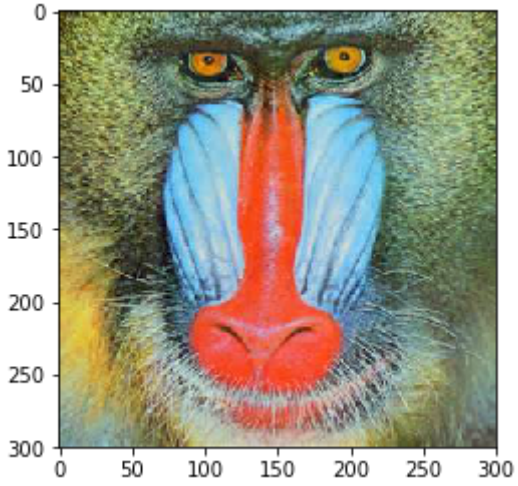
In [264]:
```
img1 = img1.astype(float)
img2 = img2.astype(float)

print(type(np.max(img1[:, :, 0])))
print(type(np.max(img2[:, :, 0])))
```

```
<class 'numpy.float64'>
<class 'numpy.float64'>
```

In [265]:
```
# normalize per channel :
img1[:, :, 0] = (img1[:, :, 0] - np.min(img1[:, :, 0])) / (np.max(img1
[:, :, 0]) - np.min(img1[:, :, 0]))
img1[:, :, 1] = (img1[:, :, 1] - np.min(img1[:, :, 1])) / (np.max(img1
[:, :, 1]) - np.min(img1[:, :, 1]))
img1[:, :, 2] = (img1[:, :, 2] - np.min(img1[:, :, 2])) / (np.max(img1
[:, :, 2]) - np.min(img1[:, :, 2]))

plt.imshow(img1)
plt.show()
```



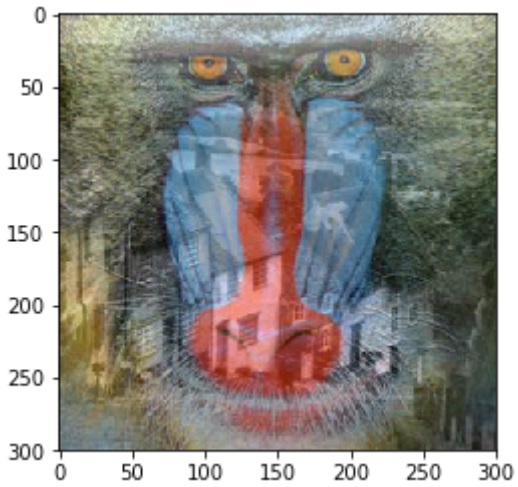
In [266]:
```
# normalize per channel :
img2[:, :, 0] = (img2[:, :, 0] - np.min(img2[:, :, 0])) / (np.max(img2
[:, :, 0]) - np.min(img2[:, :, 0]))
img2[:, :, 1] = (img2[:, :, 1] - np.min(img2[:, :, 1])) / (np.max(img2
[:, :, 1]) - np.min(img2[:, :, 1]))
img2[:, :, 2] = (img2[:, :, 2] - np.min(img2[:, :, 2])) / (np.max(img2
[:, :, 2]) - np.min(img2[:, :, 2]))

plt.imshow(img2)
plt.show()
```

In [267]:
```
#3c dd the images together and re-normalize them to have minimum value 0
 and maximum value 1.
#Display this image in your report.
```

In [268]:
```
img12 = img1 + img2
# normalize per channel :
img12[:, :, 0] = (img12[:, :, 0] - np.min(img12[:, :, 0])) / (np.max(img
12[:, :, 0]) - np.min(img12[:, :, 0]))
img12[:, :, 1] = (img12[:, :, 1] - np.min(img12[:, :, 1])) / (np.max(img
12[:, :, 1]) - np.min(img12[:, :, 1]))
img12[:, :, 2] = (img12[:, :, 2] - np.min(img12[:, :, 2])) / (np.max(img
12[:, :, 2]) - np.min(img12[:, :, 2]))

plt.imshow(img12)
plt.show()
```



In [269]:
```
# ===== Problem 3d =====
# Create a new image such that the left half of
# the image is the left half of image1 and the
# right half of the image is the right half of image2.

newImage1 = None
newImage1 = img12.copy()
half_cols = np.shape(newImage1)[1]//2
newImage1[:, :half_cols, :] = img1[:, :half_cols, :]
newImage1[:, half_cols:, :] = img2[:, half_cols:, :]
```

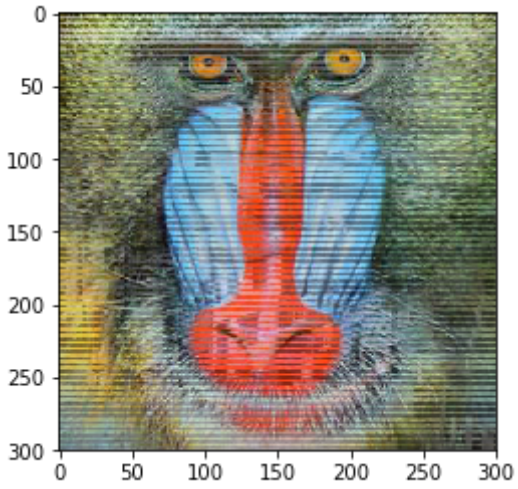
In [270]:
```
plt.imshow(newImage1)
plt.show()
```

In [271]:
```
# ===== Problem 3e =====
# Using a for loop, create a new image such that every odd
# numbered row is the corresponding row from image1 and the
# every even row is the corresponding row from image2.
# Hint: Remember that indices start at 0 and not 1 in Python.

newImage2 = None
```

In [272]:
```
newImage2 = img12.copy()
for i in range(0, np.shape(newImage2)[0], 2):
    newImage2[i,:, :] = img1[i, :, :]
for i in range(1, np.shape(img_for)[0], 2):
    newImage2[i,:, :] = newImage2[i, :, :]
```

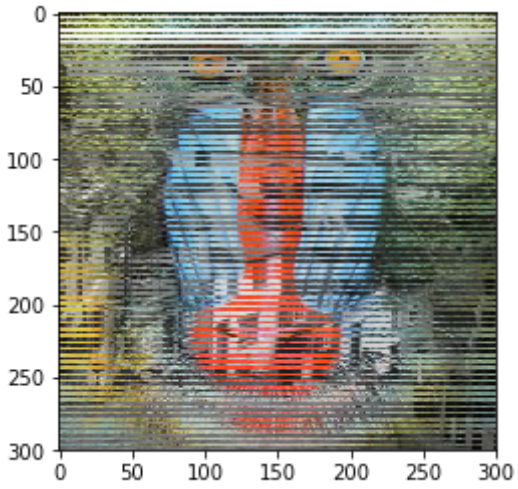
In [273]:
```
plt.imshow(newImage2)
plt.show()
```



In [274]:
```
# ===== Problem 3f =====
# Accomplish the same task as part e without using a for-loop.
# The functions reshape and repmat may be helpful here.
```

In [275]:
```
newImage3 = img12.copy()
newImage3[::2, :, :] = img1[::2, :, :]
newImage3[1::2, :, :] = img2[1::2, :, :]
```

In [276]:
```
plt.imshow(newImage3)
plt.show()
```

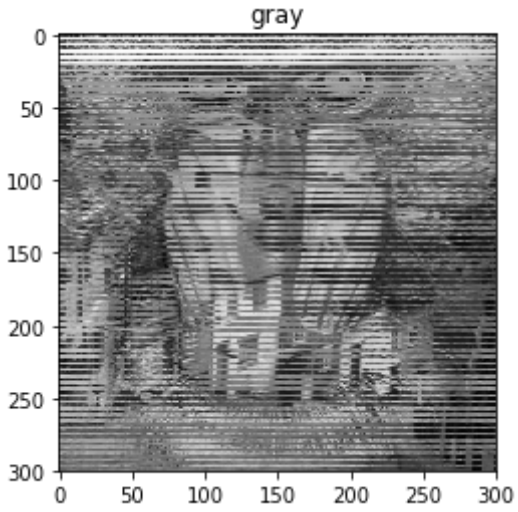


In [277]:
```
# ===== Problem 3g =====
# Convert the result from part f to a grayscale image.
# Display the grayscale image with a title.
```

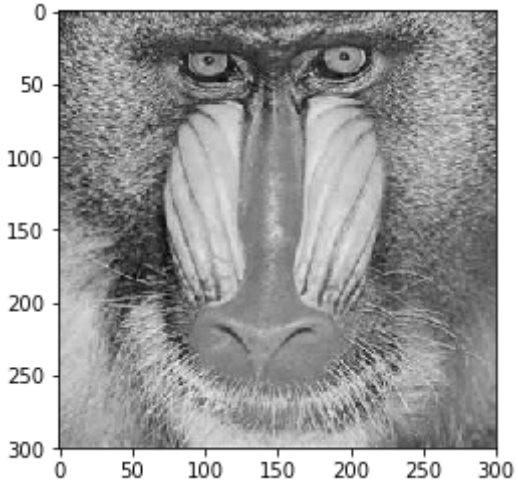
In [278]:
```
def rgb2gray(rgb):
    return np.dot(newImage3[...,:3], [0.299, 0.587, 0.114])

gray = rgb2gray(newImage3)
```

In [279]:
```
plt.imshow(gray, cmap = plt.get_cmap('gray'))
plt.title('gray')
plt.show()
```



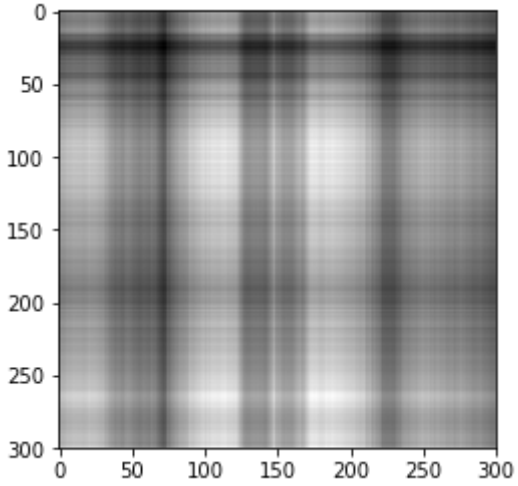
In [280]:
```
# Read in image1 as a grayscale image. Take the singular value
# decomposition of the image.
img1 = misc.imread('image1.jpg', flatten=True)
plt.imshow(img1, cmap = plt.get_cmap('gray'))
plt.show()
```



In [281]:
```
# ===== Problem 4b =====
# Save and display the best rank 1 approximation
# of the (grayscale) image1.

rank1approx = None
U, S, V_transpose = np.linalg.svd(img1)
```

In [282]:
```
rank1approx = np.matrix(U[:, :1]) * np.diag(S[:1]) * np.matrix(V_transpo
se[:1, :])
plt.imshow(rank1approx, cmap='gray');
plt.show()
```

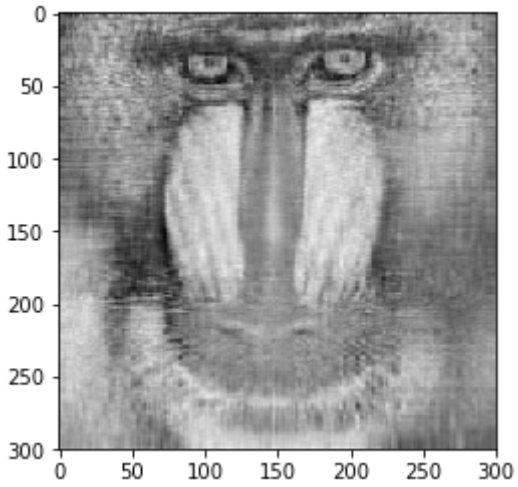


In [283]:
```
misc.imsave('rank1.png', rank1approx)
```

In [284]:
```
# ===== Problem 4c =====
# Save and display the best rank 20 approximation
# of the (grayscale) image1.

rank20approx = None
```

In [285]:
```
rank20approx = np.matrix(U[:, :20]) * np.diag(S[:20]) * np.matrix(V_tran
spose[:20, :])
plt.imshow(rank20approx, cmap='gray');
plt.show()
```



In [286]:
```
misc.imsave('rank20.png', rank20approx)
```