19AIE303 - Signal & Image Processing
Assignment - 2

CB.EN.U4AIE19059
Shiva Rupan S

1) Image compression using Wavelets

The functions 'firstthres' and 'thresecnds' are analogous to soft and hard wavelet thresholding respectively.

In [1]:
```
import numpy as NP
import matplotlib.pyplot as PLT

import pywt

from skimage import color, io
from skimage.transform import rescale, resize
```

In [2]:
```
def firstthres(c, thres):
    y = NP.floor(NP.abs(c/thres)) * NP.sign(c)
    y = NP.sign(y) * (NP.abs(y) + 0.5) * thres
    return y
```

In [3]:

```
def my_plot(y):
    # h -> Histogram values
    [h, bin_pos] = NP.histogram(y)
    m = NP.max(y)
    l = len(h)
    t = NP.linspace(-m,m,l)
    return (t,h)
```

In [4]:

```
def thresecnds(c,thres):
    return NP.floor(NP.abs(c/thres))
```

In [5]:

```
def my_plot(y,t0):
    t = NP.append(NP.arange(1/t0), NP.array(1/t0))
    h = NP.histogram(y,t)
    h = h/sum(h)
    return (t,h)
```

In [6]:
```
x = io.imread("Downloads\\Lena_image.png")
```

In [11]:
```
x = resize(x, (256,256))
x = color.rgb2gray(x)
PLT.imshow(x, cmap = 'gray')
PLT.title('Original Image')
```

```
<ipython-input-11-da862de785c1>:2: FutureWarning: The behavi
grayscale image to be passed as inputs and leaves them unmod
images with 3 channels.
  x = color.rgb2gray(x)


Text(0.5, 1.0, 'Original Image')
```

In [23]:
```
# Decomposition performed upto 4 levels ...
coeffs = pywt.wavedec2( x, 'haar', level = 4)

# 'coeffs' is of the form [cA4, (cH4,cV4,cD4), (cH3,cV3,cD3), ... (cH1,cV1,cD1)]
# cHi, cVi, cDi ---> Horizontal, vertical and diagonal coeffs. of 'i'th level
[cAn, (cH4,cV4,cD4), (cH3,cV3,cD3), (cH2,cV2,cD2), (cH1,cV1,cD1)] = coeffs
```

In [24]:
```
# cAn is the approximation coefficient at the 4th level ...
cAn.shape
```

```
(16, 16)
```

In [15]:
```
# Thresholding value ...
thres = 0.1

# Soft thresholding ...
y = pywt.threshold(cAn, thres, 'soft')
```

In [16]:
```
y.shape
```

```
(16, 16)
```

In [55]:
```
# Coefficients used for reconstruction ...
coeffs_new = [y, (cH4,cV4,cD4), (cH3,cV3,cD3), (cH2,cV2,cD2), (cH1,cV1,cD1)]
```

In [56]:
```
f = pywt.waverec2( coeffs_new , 'haar')
```

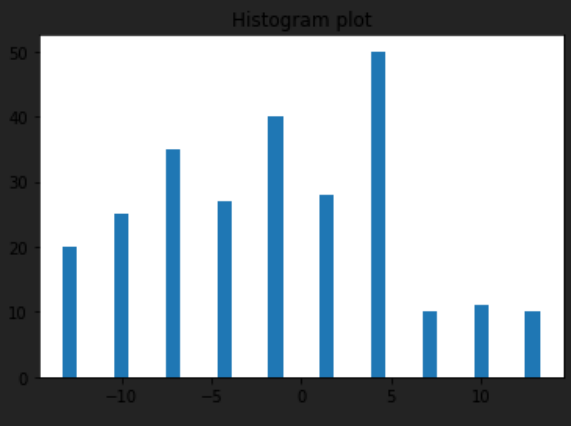
In [57]:
```
PLT.imshow(f, cmap = 'gray')
PLT.title('Reconstructed image')
```

```
Text(0.5, 1.0, 'Reconstructed image')
```

In [63]:
```
[t,h] = my_plot(y)
```

In [64]:
```
PLT.bar(t,h)
PLT.title('Histogram plot')
```

```
Text(0.5, 1.0, 'Histogram plot')
```

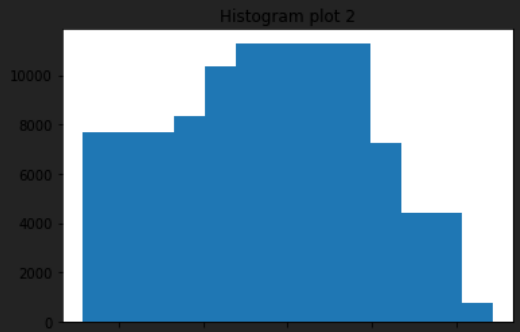


In [65]:
```
x1 = x.flatten('F')
y1 = pywt.threshold(x1, thres, 'soft')

I = y1.reshape(256,256).T
PLT.imshow(I, cmap = 'gray')
PLT.title('Reconstructed image 2')
```

```
Text(0.5, 1.0, 'Reconstructed image 2')
```

In [66]:
```
[t1,h1] = my_plot(y1)
PLT.bar(t1,h1)
PLT.title('Histogram plot 2')
```

```
Text(0.5, 1.0, 'Histogram plot 2')
```



In [69]:
```
coeffs1 = pywt.wavedec2(x, "haar" , level=4)
[cAn1, (cH4,cV4,cD4), (cH3,cV3,cD3), (cH2,cV2,cD2), (cH1,cV1,cD1)] = coeffs1
```

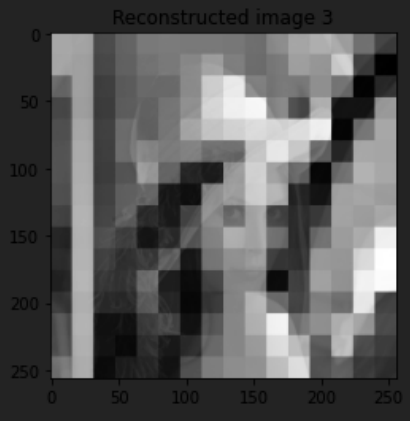
In [70]:
```
t0 = 0.089
t = NP.arange(1/t0)
```

In [71]:
```
y2 = thresecnds(cAn1,t0)
```

In [72]:
```
coeffs1_new = [y2, (cH4,cV4,cD4), (cH3,cV3,cD3), (cH2,cV2,cD2), (cH1,cV1,cD1)]
```

In [73]:
```
f1 = pywt.waverec2(coeffs1_new, 'haar')
PLT.imshow(f1, cmap = 'gray')
PLT.title('Reconstructed image 3')
```

```
Text(0.5, 1.0, 'Reconstructed image 3')
```



```
In [86]:
x1 = x.flatten('F')
t0 = 0.089
t = NP.arange(1/t0)
y2 = thresecnds(x1,t0)
```

```
In [87]:
I_new = y2.reshape(256,256).T
PLT.imshow(I_new, cmap = 'gray')
PLT.title('Reconstructed image 4')
```

```
Text(0.5, 1.0, 'Reconstructed image 4')
```

---

2)

a) **<u>Image compression using SVD</u>**

<u>Python code:</u>

In [1]:
```
import numpy as NP
import matplotlib.pyplot as PLT

from skimage import io
from scipy.linalg import svd
```

In [2]:
```
x = io.imread("Downloads\\cameraman.tif")
x.shape
```

```
(256, 256)
```

In [3]:
```
# im2double in MATLAB converts the intensity values to the range 0-1
# Here, the same is performed with x/255
x = NP.array( x, dtype = 'float64')
x = x/255
```

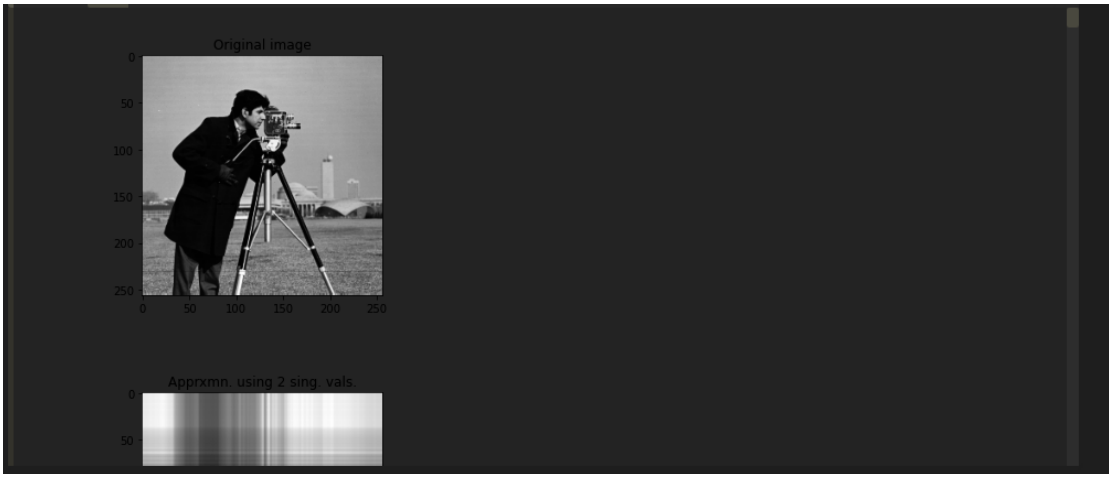
In [4]:
```
[m,n] = x.shape
```

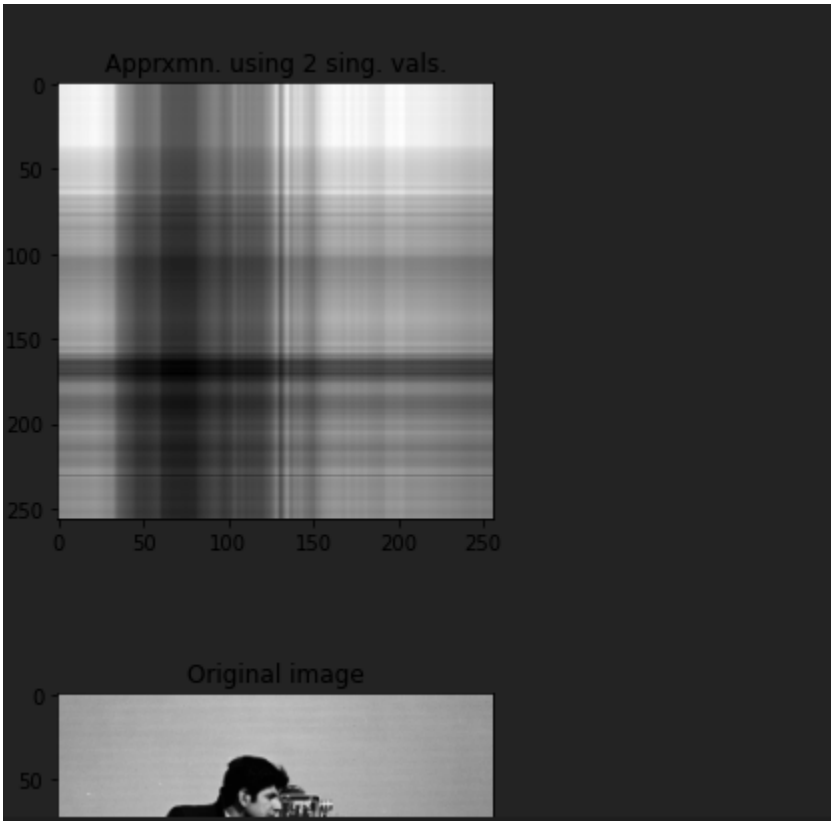
In [5]:
```
# NOTE: 's' is a vector holding all the singular values. It is not a diagonal matrix
u,s,vT = svd(x)
```

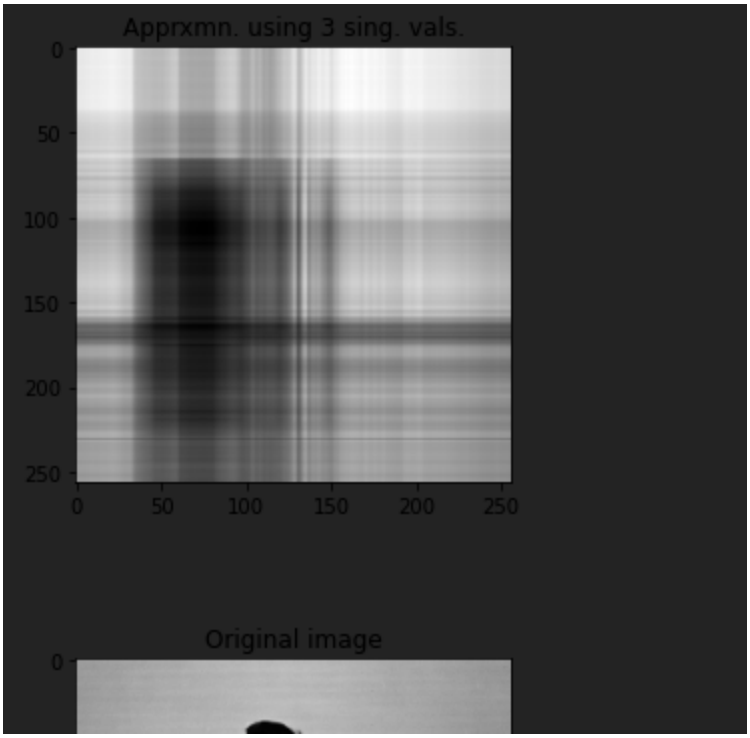
In [6]:
```
# 'A' will finally hold the reconstructed image obtained after omitting
# certain singular values
A = NP.zeros([m,n],'float64')
```

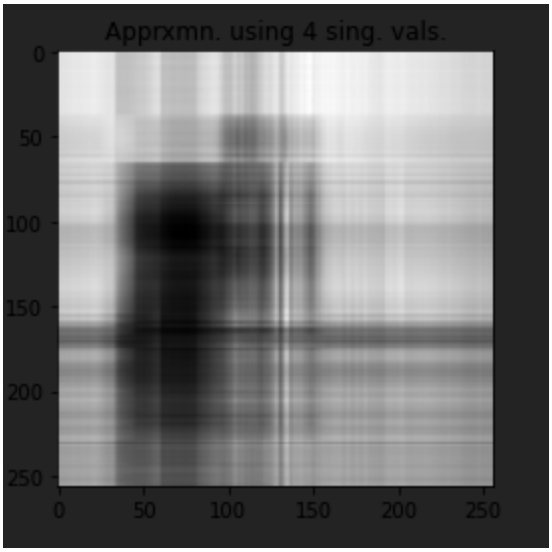
In [235]:
```
for i in range(1,50):

    # NP.matrix(u[:, :i]) ---> First 'i' columns in 'u'
    # NP.diag(s[:i]) ---> Diagonal matrix with first 'i' singular values lying along the diagonal
    # NP.matrix(vT[:i, :]) ---> First 'i' rows of 'vT'

    A = A +  NP.matrix(u[:, :i]) * NP.diag(s[:i]) * NP.matrix(vT[:i, :])

    PLT.imshow( x, cmap = 'gray')
    PLT.title('Original image')
    PLT.show()

    PLT.imshow( A, cmap = 'gray')
    PLT.title('Apprxmn. using {0} sing. vals.'.format(i+1))
    PLT.show()
```

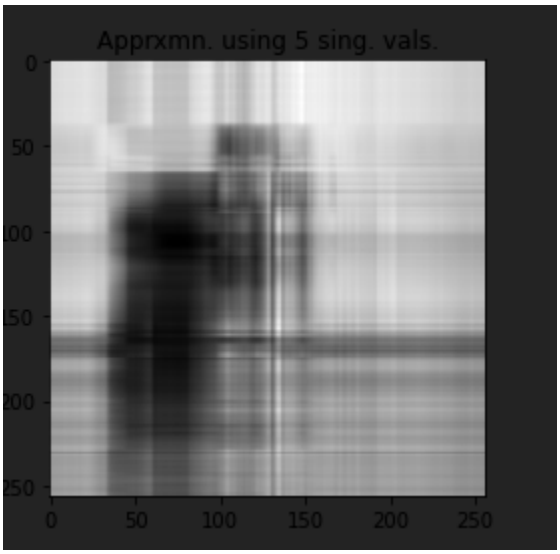
Output :

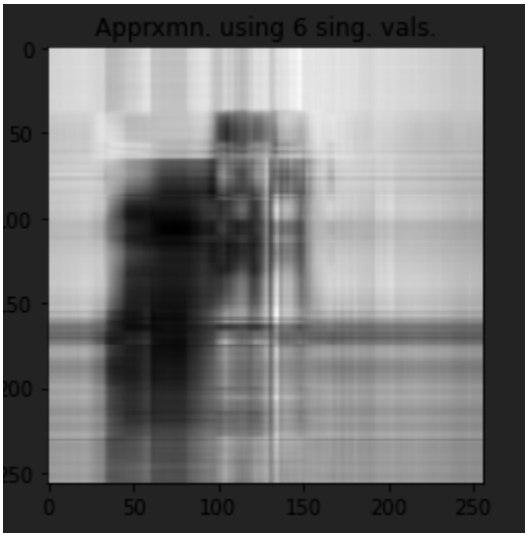

Original image
0
50
100
150
200
250
0
50
100
150
200
250
Apprxmn. using 2 sing. vals.
0
50

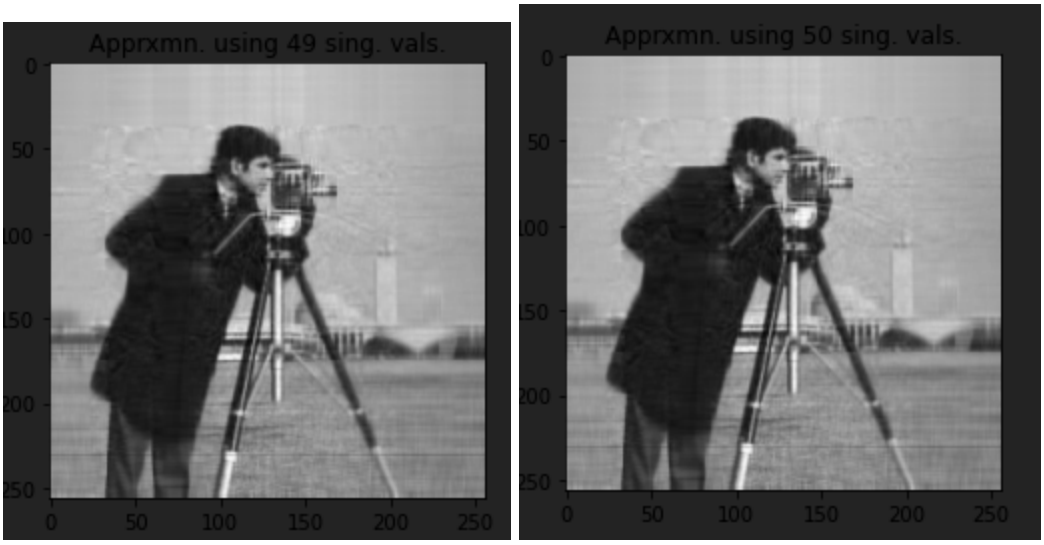

Apprxmn. using 2 sing. vals.
0
50
100
150
200
250
0
50
100
150
200
250
Original image
0
50


Apprxmn. using 3 sing. vals.
0
50
100
150
200
250
0
50
100
150
200
250
Original image
0


Apprxmn. using 4 sing. vals.
0
50
100
150
200
250
0
50
100
150
200
250


Apprxmn. using 5 sing. vals.
0
50
100
150
200
250
0
50
100
150
200
250


Apprxmn. using 6 sing. vals.
0
50
100
150
200
250
0
50
100
150
200
250



---

b) Image watermarking using SVD

Python code:

```
In [1]:
import numpy as NP
import matplotlib.pyplot as PLT

from skimage import io, color
from skimage.transform import resize
from scipy.linalg import svd

In [2]:
# Cover Image ...

x = io.imread("Downloads\\cameraman.tif")
# Converting to 'float64' from 'uint8'
x = NP.array(x, dtype = 'float64')
# im2double in MATLAB converts to the range 0-1. The same is achieved here using x/255
x = x/255
```

In [3]:
```
PLT.imshow(x, cmap = 'gray')
```

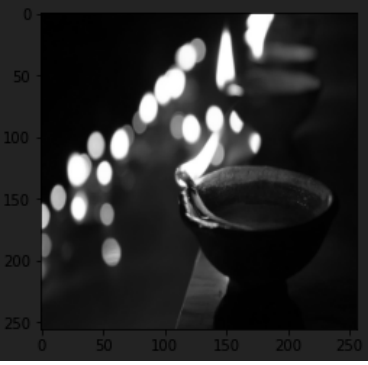
```
<matplotlib.image.AxesImage at 0x1e4a963bc40>
```

In [4]:
```
x.shape
```

```
(256, 256)
```

In [5]:
```
# Image to be hidden ...

w1 = io.imread("Downloads\\Deepam.jpg")

# NOTE: rgb2gray automatically converts to the range 0-1. So, w1/255 is not needed
w1 = color.rgb2gray(w1)
w = resize(w1, [256,256])
PLT.imshow(w, cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x1e4a99a8760>
```



In [6]:
```
w.shape
```

```
(256, 256)
```

In [7]:
```
[u,s,v] = svd(x)
```

In [8]:
```
# 's' is a vector containing singular values ...
# 'ss' has those singular values along its diagonal ...
ss = NP.diag(s)
```

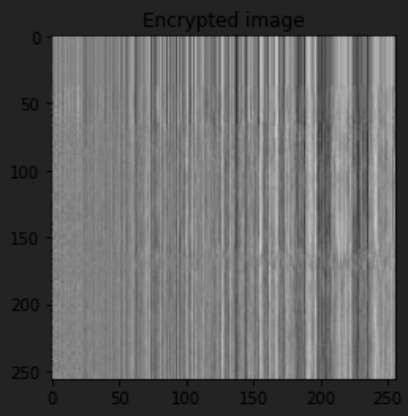
In [9]:
```
alpha = 0.2
```

In [10]:
```
# Scaling the image to be hidden by a factor of 'alpha' and adding it to the
# matrix of singular values
s = NP.diag(s) + alpha*w
```

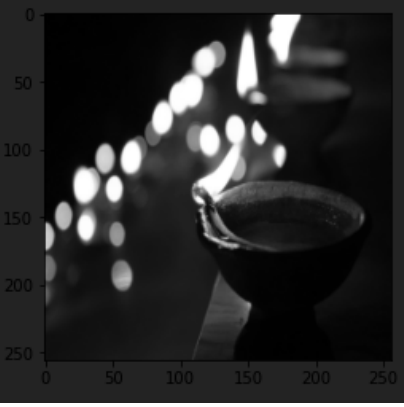
In [11]:
```
# Aencrypt = u * s * transpose(v)
Aencrypt = u.dot(s).dot(v.T)
```

In [12]:
```
PLT.imshow(Aencrypt, cmap = 'gray')
PLT.title("Encrypted image")
```

```
Text(0.5, 1.0, 'Encrypted image')
```



```
In [13]:  # SVD of the encrypted image
          [ud, sd, vd] = svd(Aencrypt)

In [14]:  ssd = sd

In [23]:  # (1/alpha) * (transpose(u)*Aencrypt*v) - (1/alpha) * ('s' matrix of Cover Image)
          Recovered = (1/alpha)*NP.matmul(NP.matmul(u.T, Aencrypt), v) - (1/alpha)*ss

In [25]:  PLT.imshow(Recovered, cmap = 'gray')

          <matplotlib.image.AxesImage at 0x1e4aa9a8f10>
```



---

c) **Image fusion using SVD**

Python code:

```
In [4]:  from skimage import io, color
         import matplotlib.pyplot as PLT
         import numpy as NP

In [2]:  A = io.imread("Downloads\\Time1.bmp")
         B = io.imread("Downloads\\Time2.bmp")

         A = color.rgb2gray(A)
         B = color.rgb2gray(B)
```

In [5]:
```
PLT.imshow(A, cmap = 'gray')
PLT.title("Time1.bmp")
```

```
Text(0.5, 1.0, 'Time1.bmp')
```

In [6]:
```
PLT.imshow(B, cmap = 'gray')
PLT.title("Time2.bmp")
```

```
Text(0.5, 1.0, 'Time2.bmp')
```

In [7]:
```
A.shape
```

```
(480, 640)
```

In [8]:
```
# 'flatten' coverts A from (480,640) to (480x640,1), i.e., (307200,1)
A_flt = A.flatten('F').T

# Converting 'B' from (480,640) to (307200,1)
B_flt = B.flatten('F').T

# Concatenating 'A_flt' and 'B_flt' along axis-1 (columns) to produce 'C'
# of shape (307200,2)
C = NP.concatenate((NP.array([A_flt]).T, NP.array([B_flt]).T), axis = 1)
```

In [9]:
```
C.shape
```

```
(307200, 2)
```

In [10]:
```
N = len(A_flt)

mA = NP.mean(A_flt) # Mean(A_flt)
mB = NP.mean(B_flt) # Mean(B_flt)
```

In [11]:
```
# (1/(N-1)) * (A_flt - mA) * (A_flt - mA)
CAA = (1/(N-1)) * NP.sum(NP.multiply(A_flt - mA, A_flt - mA))

# (1/(N-1)) * (A_flt - mA) * (B_flt - mB)
CAB = (1/(N-1)) * NP.sum(NP.multiply(A_flt - mA, B_flt - mB))

# (1/(N-1)) * (B_flt - mB) * (A_flt - mA)
CBA = (1/(N-1)) * NP.sum(NP.multiply(B_flt - mB, A_flt - mA))

# (1/(N-1)) * (B_flt - mB) * (B_flt - mB)
CBB = (1/(N-1)) * NP.sum(NP.multiply(B_flt - mB, B_flt - mB))
```

In [12]:
```
D = NP.array([
              [CAA, CAB],
              [CBA, CBB]
             ])
```

In [13]:
```
[evalues, evectors] = NP.linalg.eig(D)
```

In [14]:
```
evalues
```

```
array([0.00159812, 0.06355801])
```

In [15]:
```
# Eigen vectors are stored column-wise ...
evectors
```

```
array([[-0.71378485, -0.70036504],
       [ 0.70036504, -0.71378485]])
```

In [16]:
```
i = 0

if (evalues[i] > evalues[i+1]):
    vect = evectors[:,i]
else:
    vect = evectors[:,i+1]
```

In [17]:
```
vect
```

```
array([-0.70036504, -0.71378485])
```

In [18]:
```
# Fusing the two images ...
ImgFus = (vect[0]*A + vect[1]*B)/NP.sum(vect)
```

In [19]:
```
ImgFus.shape
```

```
(480, 640)
```

```
In [20]: PLT.imshow(ImgFus, cmap = 'gray')
         PLT.title("Fusion of Time1.bmp and Time2.bmp")
```

```
Text(0.5, 1.0, 'Fusion of Time1.bmp and Time2.bmp')
```

~~~~~~

==
~~~~~~