final question 3 decompression

March 25, 2018

In [1]:
```
# import the packages
import numpy as np
from scipy.misc import imread, imresize, imsave
import matplotlib.pyplot as plt
from numpy import matlib
import math
from scipy import stats
import imageio
from skimage.transform import resize
import skimage
import zlib, sys
import gzip
import matplotlib
import scipy
import copy
import random
import numpy
import sympy as sp
```

In [2]:
```
# define a function to covert the image to a gray scale image
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

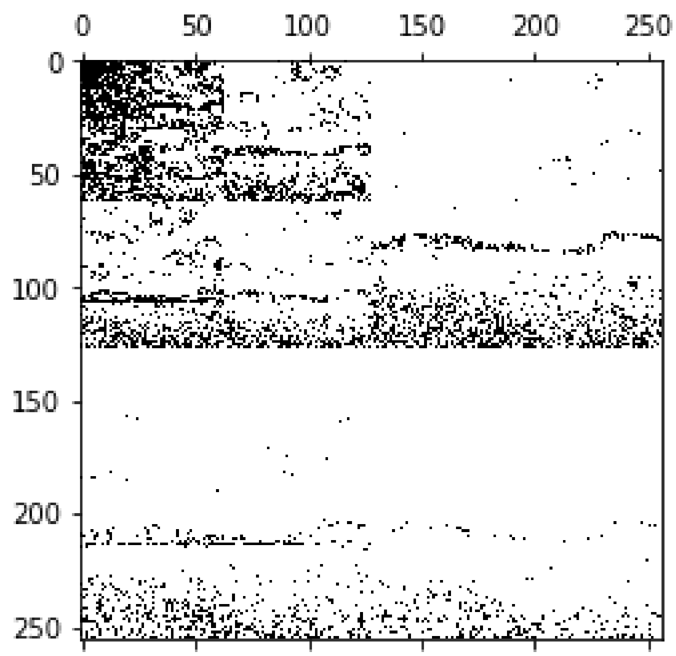
In [3]:
```
L0=0.99436891104358
L1=0.41984465132951
L2=-0.17677669529664
L3=-0.06629126073624
L4=0.03314563036812

H0=-0.70710678118655
H1=0.35355339059327
H2=0
H3=0
H4=0

def CDF(N):
    TA=np.zeros((N,N+8))
```

```
for i in range(0,N,2):
    TA[i][i]=L4
    TA[i][i+1]=L3
    TA[i][i+2]=L2
    TA[i][i+3]=L1
    TA[i][i+4]=L0
    TA[i][i+5]=L1
    TA[i][i+6]=L2
    TA[i][i+7]=L3
    TA[i][i+8]=L4

for i in range(1,N,2):
    TA[i][i]=H4
    TA[i][i+1]=H3
    TA[i][i+2]=H2
    TA[i][i+3]=H1
    TA[i][i+4]=H0
    TA[i][i+5]=H1
    TA[i][i+6]=H2
    TA[i][i+7]=H3
    TA[i][i+8]=H4

TA[0][4]=L0
TA[0][5]=2*L1
TA[0][6]=2*L2
TA[0][7]=2*L3
TA[0][8]=2*L4

TA[1][4]=H1
TA[1][5]=H0+H2
TA[1][6]=H1+H3
TA[1][7]=H2+H4
TA[1][8]=H3
TA[1][9]=H4

TA[2][4]=L2
TA[2][5]=L1+L3
TA[2][6]=L0+L4
TA[2][7]=L1
TA[2][8]=L2
TA[2][9]=L3


TA[3][4]=H3
TA[3][5]=H2+H4
TA[3][6]=H1
TA[3][7]=H0
```

```
    TA[3][8]=H1
    TA[3][9]=H2

    TA[N-1][N+8-5]=H0
    TA[N-1][N+8-6]=2*H1
    TA[N-1][N+8-7]=2*H2
    TA[N-1][N+8-8]=2*H3
    TA[N-1][N+8-9]=2*H4

    TA[N-2][N+8-5]=L1
    TA[N-2][N+8-6]=L0+L2
    TA[N-2][N+8-7]=L1+L3
    TA[N-2][N+8-8]=L2+L4
    TA[N-2][N+8-9]=L3
    TA[N-2][N+8-10]=L4

    TA[N-3][N+8-5]=H2
    TA[N-3][N+8-6]=H1+H3
    TA[N-3][N+8-7]=H0+H4
    TA[N-3][N+8-8]=H1
    TA[N-3][N+8-9]=H2
    TA[N-3][N+8-10]=H3

    TA[N-4][N+8-5]=L3
    TA[N-4][N+8-6]=L2+L4
    TA[N-4][N+8-7]=L1
    TA[N-4][N+8-8]=L0
    TA[N-4][N+8-9]=L1
    TA[N-4][N+8-10]=L2

    TA=TA[:, 4:N+8-4]
    TS=np.linalg.inv(TA)

    P = np.vstack((matlib.eye(N)[::2,:],matlib.eye(N)[1::2,:]))
    return TS,P
```

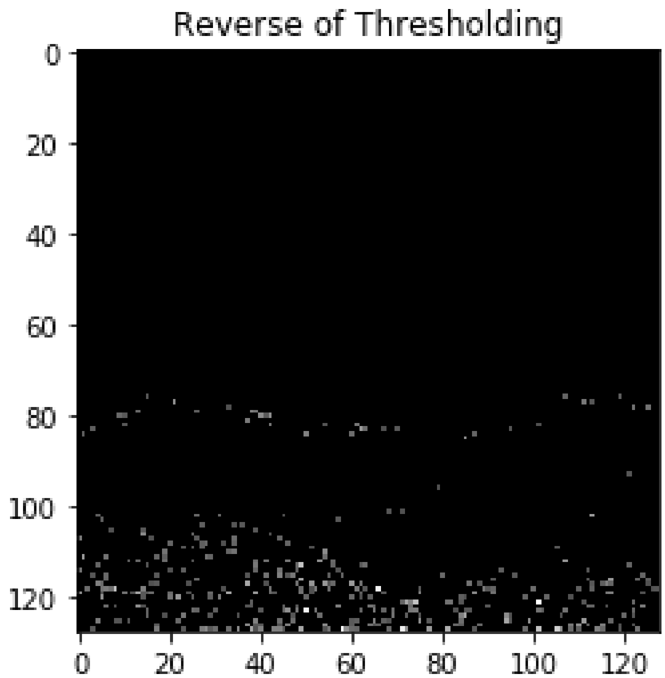
In [4]:
```
# use zlib to uncompress the compressed_indices
compressed_indices= gzip.open('compressed_indices.txt.gz', 'rb').read()
decompress_indices = zlib.decompress(compressed_indices)

# convert the byte-like object to numpy array
decompress_indices = np.frombuffer(decompress_indices, dtype=int)

# reshape the data to 2D
indices = np.reshape(decompress_indices, (256, 256))

# show the image before reverse log quantization
```

```
plt.spy(indices)
plt.show()

#print(indices)

# use zlib to uncompress the compressed_indices
compressed_sign= gzip.open('compressed_sign.txt.gz', 'rb').read()
decompress_sign = zlib.decompress(compressed_sign)

# convert the byte-like object to numpy array
decompress_sign = np.frombuffer(decompress_sign, dtype=int)

# reshape the data to 2D
sign = np.reshape(decompress_sign, (256, 256))

sign=sign/4607182418800017408.0
sign=sign.astype(int)

#print(sign)
```



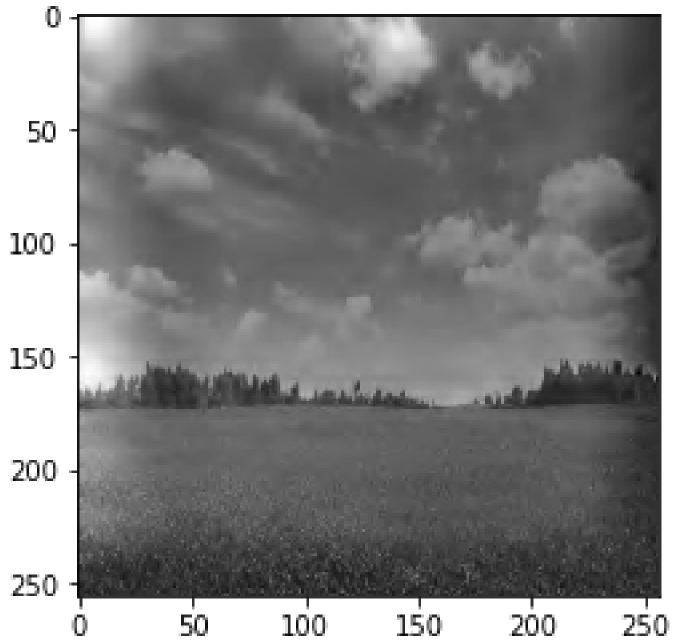
In [5]:
```
# make the codebook.txt readable for python
codebook = []

with open('codebook.txt', 'r') as f1:
    codebook = [line.strip() for line in f1]
```

```python
codebook = [float(i) for i in codebook]

#print(codebook)

# using codebook and indices to recover the quanta data
quanta = np.empty((256, 256))

for i in range(quanta.shape[0]):
    for j in range(quanta.shape[1]):
        quanta[i][j] = codebook[indices[i][j]]

#print(quanta)


# reverse threshold to F
# make a deep copy of F as G
G = copy.deepcopy(quanta)

# read in F row by row, find the min nonzero pixel
# put the number from data codebook before apply thresholding function
# in order to put the data back to nonzero
def reverse_thresholding(source):
    index = 0

    for i in range(source.shape[0]):
        for j in range(source.shape[1]):

            if source[i][j] == 0:
                index += 1
            else:
                continue


# Apply reverse thresholding function to M
reverse_thresholding(G)
#print(G)

# show the image after apply to reverse threshold
plt.imshow(G[128:256,128:256], cmap = plt.get_cmap('gray'))
plt.title("Reverse of Thresholding")
plt.show()
```



In [6]:
```python
# # open the sign.txt to put back the sign
# sign = open('sign.txt', 'rb').read()

# # convert the byte-like object to numpy array
# sign = np.frombuffer(sign)

# # reshape the sign to 2D numpy array
# sign = np.reshape(sign, (256, 256))
# #print(sign)


# put the nagative sign back to the correct position
G = G * sign
#print(G)


# make a deep copy of G
J = copy.deepcopy(G)

# get number of times of decoding and the starting point
N = len(J)
times = int(np.log2(N))-1
start = 16

# Doing full-level decoding (Backward Haar Transform)
```

```
for i in range(5):
    TS,P = CDF(start)
    J[0:start, 0:start] = TS.T*P.T*J[0:start, 0:start]*P*TS
    start = 2 * start

# show the result of full-level decoding
plt.figure()
plt.imshow(J, cmap = plt.get_cmap('gray'))
plt.show()

# print the info of J
#print(J)
```



In [7]:
```
####################
# get PSNR#
####################

# get the information from the original image(before full-level encoding)
# reads in the original image
A = imageio.imread('image.jpg')

# resize the image(before apply gray scale function) as a 256 by 256 matrix
A = skimage.transform.resize(A, [256, 256], mode='constant')

# Apply the rgb2gray function to the image
A = rgb2gray(A)
```

```python
#print(A)

# get the maximum value of the original image
maxValue = np.amax(A)
#print(maxValue)

# get the 2D info of origianl image
#print(A)

# get the 2D info of the reconstructed image
#print(J)

# calculate the MSE
MSE_arr = np.empty([J.shape[0], J.shape[1]])
for i in range(J.shape[0]):
    for j in range(J.shape[1]):
        MSE_arr[i][j] = ((A[i][j] - J[i][j])**2)

MSE = 0
for a in range(MSE_arr.shape[0]):
    for b in range(MSE_arr.shape[1]):
        MSE += MSE_arr[a][b]

MSE = MSE/(MSE_arr.shape[0]*MSE_arr.shape[1])

#print(MSE)


# calculate the PSNR
PSNR = 20*math.log10(maxValue) - 10*math.log10(MSE)
print(int(PSNR))
```

```
12
```