# final question 3 compression

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])
```

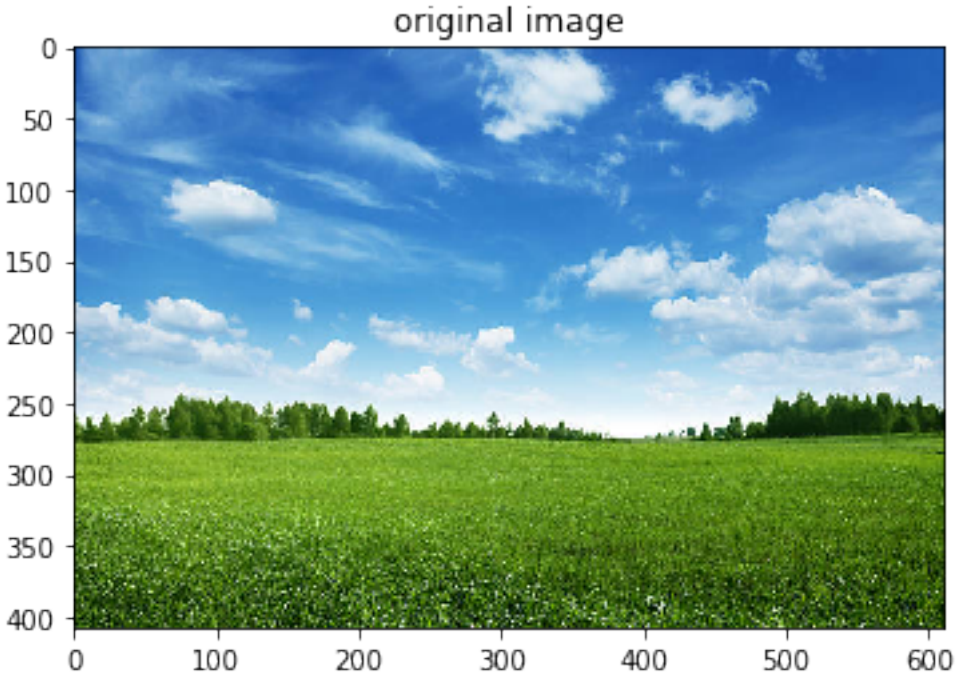
In [3]:
```
L0=0.99436891104358
L1=0.41984465132951
L2=-0.17677669529664
L3=-0.06629126073624
L4=0.03314563036812

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

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

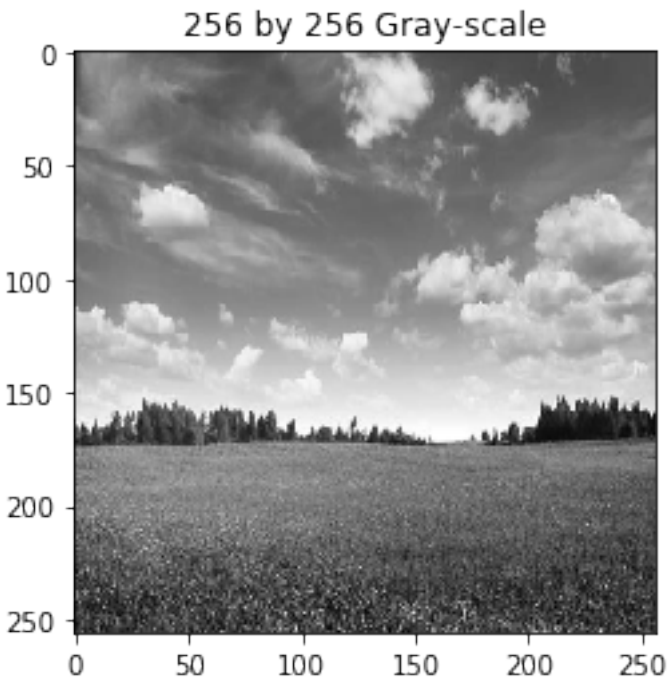
```
for i in range(0,N,2):
    TA[i][i]=L4
    TA[i][i+1]=L3
    TA[i][i+2]=L2
    TA[i][i+3]=L1
    TA[i][i+4]=L0
    TA[i][i+5]=L1
    TA[i][i+6]=L2
    TA[i][i+7]=L3
    TA[i][i+8]=L4

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]

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

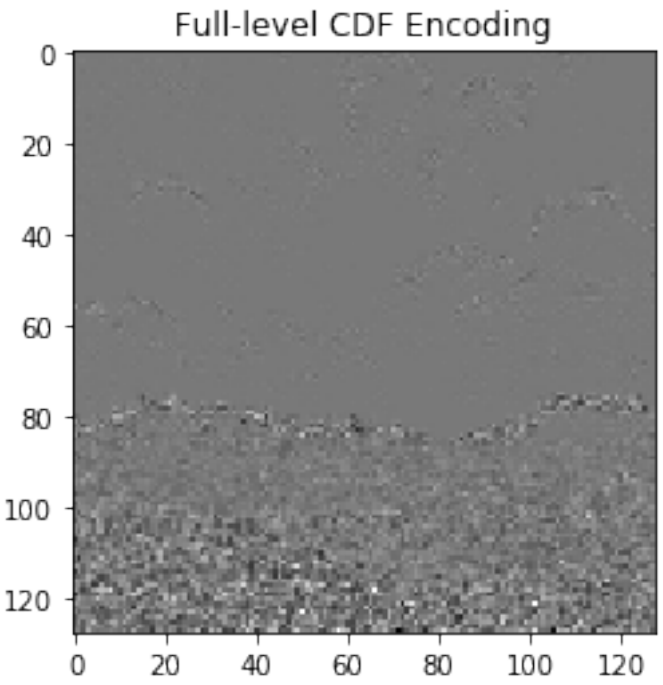
In [4]:
```
# reads in a jpeg image
A = imageio.imread('image.jpg')

# show the original image just read in
plt.imshow(A, cmap = plt.get_cmap('gray'))
plt.title("original image")
plt.show()

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

#print(A)
```

```
# show the jpeg image in a figure
# plt.imshow(A, cmap = plt.get_cmap('gray'))
# plt.title("original image after resize")
# plt.show()


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

#print(A)

# show the jpeg image in a figure
plt.imshow(A, cmap = plt.get_cmap('gray'))
plt.title("256 by 256 Gray-scale")
plt.show()
```





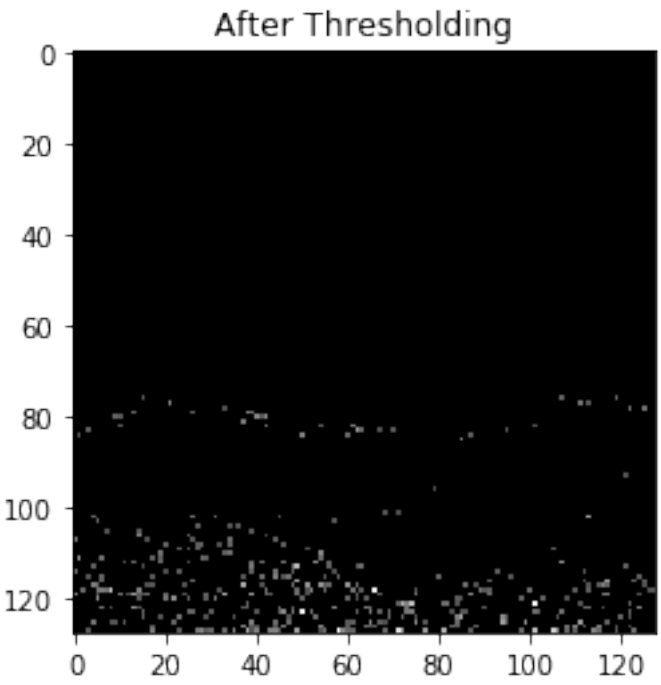

In [5]:
```
# make a deep copy of resize&gray-scale image
B = copy.deepcopy(A)

# set size to 256
N = 256

# Doing Encoding
for i in range(5):
    TA,P= CDF(N)
    #print(TA.shape)
    B[0:N, 0:N] = P*TA*B[0:N, 0:N]*TA.T*P.T
    N = int(N/2)
#     if N==4:
#         break

# show the result of full-level encoding
plt.figure()
plt.imshow(B[128:256,128:256], cmap = plt.get_cmap('gray'))
plt.title("Full-level CDF Encoding")
plt.show()

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




In [6]:
```
# create an empty numpy array record the sign of array
sign = np.empty([256,256])

# record the sign
for i in range(B.shape[0]):
    for j in range(B.shape[1]):
        if B[i][j]<=0:
            sign[i][j] = -1
        else:
            sign[i][j] = 1

#print(sign)



# make 2 deep copy of B
X = abs(copy.deepcopy(B))
Y = copy.deepcopy(B)

# convert X(2D numpy array) into 1D numpy array
Y = Y.ravel()

#print(X)
```

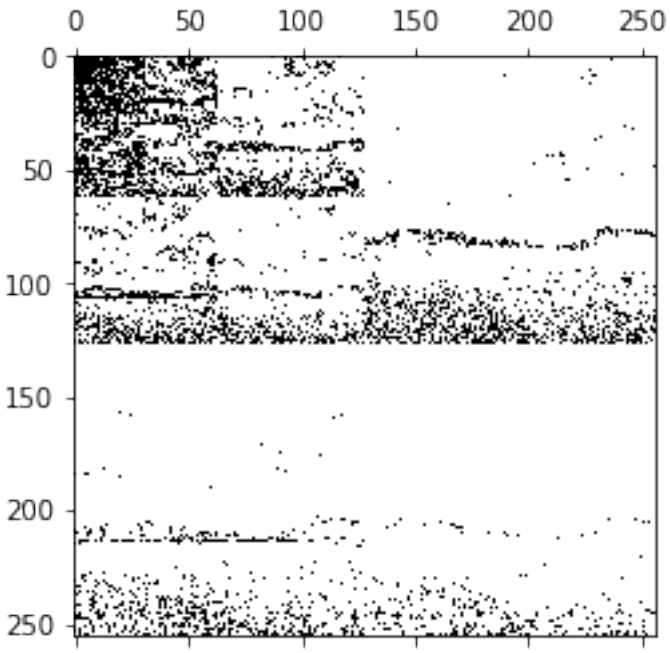
```
        # make a deep copy to X to get the threshold but not affect X
        Z = copy.deepcopy(Y)

        # sort the numpy array by its absolute value
        Z = np.sort(abs(Z))

        # promopt to ask user what the top percent pixel will retain the same
        cutoff = input('How many percents of smallest elements you want to set to zero?')

        # define thresholding function to find the threshold
        def find_th(source, percentage):
            index = 0
            index = math.floor(len(source) * percentage / 100)
            threshold = source[index]
            return threshold

        # apply the thresholding function to find the threshold th
        th = find_th(Z, int(cutoff))
        #print(th)

        # implementation of the threshold process to numpy array X
        if th!=0:
            for i in range(X.shape[0]):
                for j in range(X.shape[1]):

                    if X[i][j] > th:
                        continue
                    else:
                        X[i][j] = 0



        # show the image after apply to threshold
        plt.imshow(X[128:256,128:256], cmap = plt.get_cmap('gray'))
        plt.title("After Thresholding")
        plt.show()

        # print the matrix out the make sure A apply to the threshold function correctly
        #print(X)
```

```
How many percents of smallest elements you want to set to zero?90
```




In [7]:
```
# initialize the value to create proper partition and codebook
MX = np.amax(X)
bits = 8
NP = 2**(bits-1)-1
c1 = 0
diff = (MX/th)**(1/NP)

# create empty list of partition and codebook
partition = []
codebook = [c1]

# create partition list
for n in range(NP):
    partition.append(th*(diff**n))

# print the length of partition list
print(len(partition))

# create codebook list
for n in range(NP-1):
    codebook.append(random.uniform(partition[n], partition[n+1]))
codebook.append(random.uniform(partition[len(partition)-1], partition[len(partition)-1

# print the length of the codebook list
print(len(codebook))
```

```
# convert M(2D numpy array) into 1D list as signal
signal = []

for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        signal.append(X[i][j])


# define a function to do quantization
def quantiz(signal, partition, codebook):
    indices = []
    quanta = []

    for data in signal:
        index = 0
        while index<len(partition) and data>partition[index]:
            index += 1
        indices.append(index)
        quanta.append(codebook[index])

    return indices, quanta

# call the quantiz function to get indices and quantized signal list
indices, quanta = quantiz(signal, partition, codebook)

# reshape quantized signal into 2D array
quanta = np.reshape(quanta, (256,256))

#print(quanta)


# reshape the indices into 2D array
indices = np.reshape(indices, (256,256))
#print(indices)
#print(type(indices))

# show the image after apply to log quantization
plt.spy(indices)
plt.show()

#print(indices)
```




In [8]:
```
# make a copy of image after thresholding and log_quantiz as N
N = copy.deepcopy(indices)
Q = copy.deepcopy(sign)

A = imread('image.jpg') ;
A = rgb2gray(A);
A=imresize(A,[256,256],interp='bicubic')

#print(sign)

compressed_indices = zlib.compress(N, 9)
compressed_sign = zlib.compress(Q, 9)

# start of the lossless compression by using zlib
compress_ratio = sys.getsizeof(A)/(float(sys.getsizeof(compressed_sign))
                                   +sys.getsizeof(compressed_indices))
print("compress_ratio:", compress_ratio )
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launche
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  """
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launche
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``skimage.transform.resize`` instead.
```

```
  import sys
```

```
compress_ratio: 2.3560149296583406
```

```
In [9]: # create a file to save the compressed data
        f = gzip.open('compressed_indices.txt.gz', 'wb')
        f.write(compressed_indices)
        f.close()

        # create a file to save the sign np array
        f2 = gzip.open('compressed_sign.txt.gz', 'wb+')
        f2.write(compressed_sign)
        f2.close()

        # create a file to save the codebook list
        f1 = open('codebook.txt', 'w')
        for item in codebook:
            f1.write("%s\n" % item)
        f1.close()
```