**Mohit Mohit**

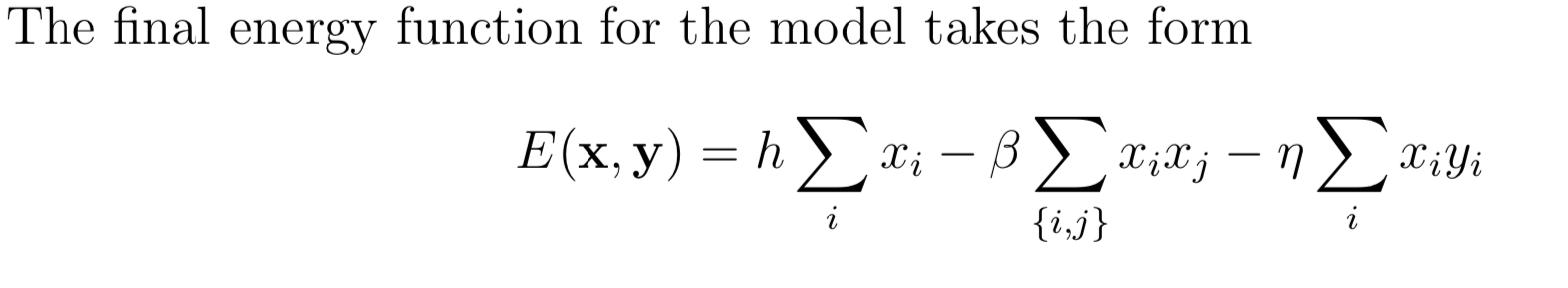
**(U33745431)**

**PROGRAMMING REPORT**

**Image Denoising**

The goal of this problem is to recover the clean image from a noisy input. The graphical model we use is a pairwise MRF. A pixel has four neighbors.

We are going to use Markov Random Fields (MRFs) to model the distribution of natural images and restore the clean image given a noisy input image.



# Solution:

To denoise the image we need to use the given energy function and initially need to guess the values for *h*, *beta* and *eta.*

After experimentation I have finalized the values as below:

h = 0.1

beta = 2

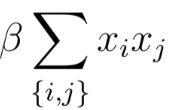
eta = 0.1

Given image is a binary image where pixel values belongs to {1, -1 }.

## Lambda function :

**neighbors**  will return all the neighbors of a pixel.

## energy function:

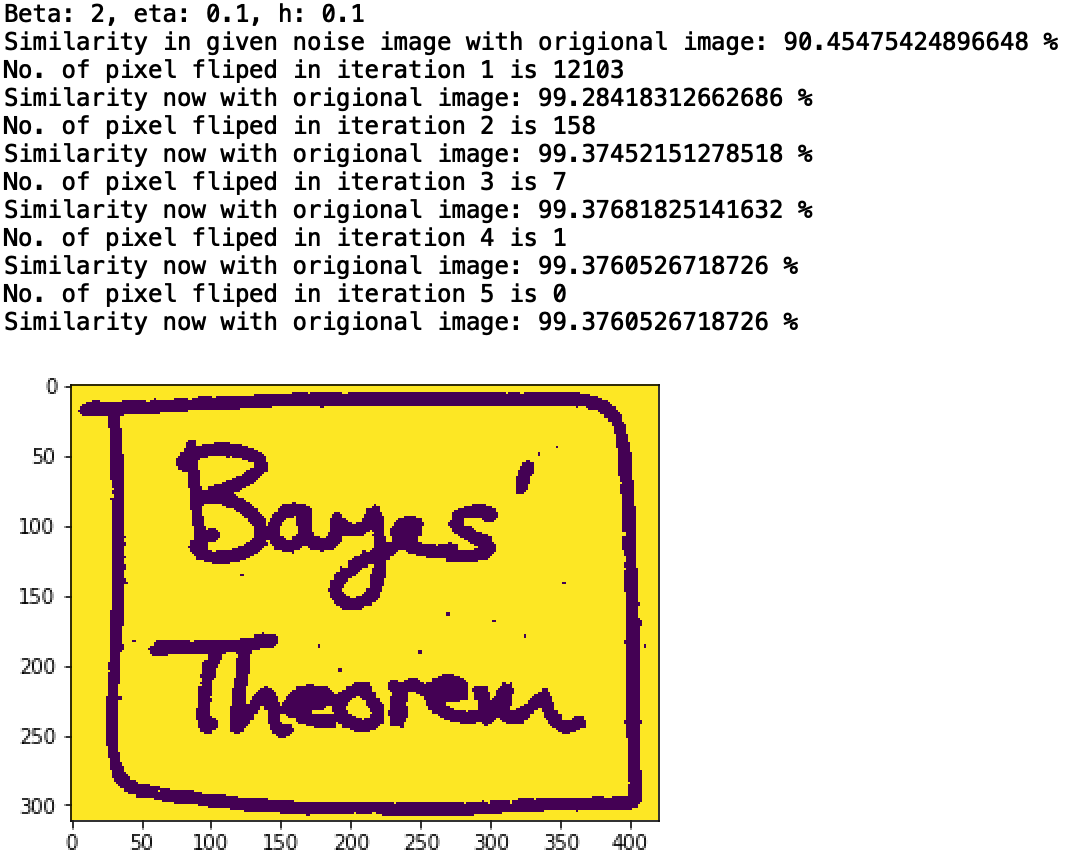
will call **summation function** to get the value of  and will return the energy for the given pixel.

Now for each pixel in the image I am calling the energy function twice. 1st time with original value of the pixel and 2nd time for the flipped value of the pixel. After getting both energies we will choose the pixel value which gives the low energy.

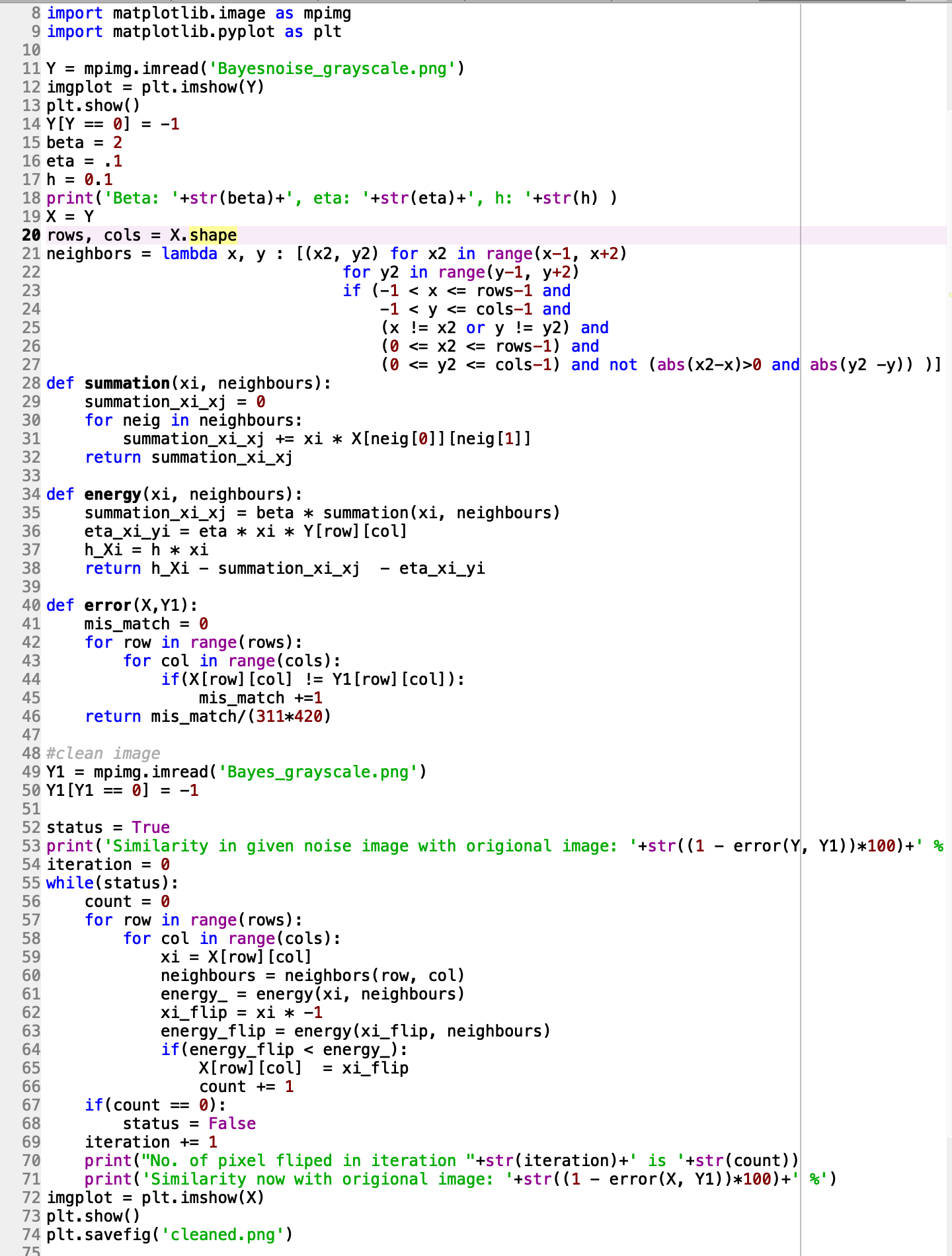
Keep doing this for the image again and again until there is 0 number of flips in a full iteration.

For the below given code we are able to get the denoised image with 99.38 similarity the original clean image. (Converges after 5 Iteration as given in the below screen shot)

Result:-



Code:



﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Apr 17 18:50:32 2019

@author: mohitbeniwal

"""

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

Y = mpimg.imread('Bayesnoise\_grayscale.png')

imgplot = plt.imshow(Y)

Y[Y == 0] = -1

beta = 2

eta = .1

h = 0.1

print('Beta: '+str(beta)+', eta: '+str(eta)+', h: '+str(h) )

X = Y

rows, cols = X.shape

neighbors = lambda x, y : [(x2, y2) for x2 in range(x-1, x+2)

for y2 in range(y-1, y+2)

if (-1 < x <= rows-1 and

-1 < y <= cols-1 and

(x != x2 or y != y2) and

(0 <= x2 <= rows-1) and

(0 <= y2 <= cols-1) and not (abs(x2-x)>0 and abs(y2 -y)) )]

def summation(xi, neighbours):

summation\_xi\_xj = 0

for neig in neighbours:

summation\_xi\_xj += xi \* X[neig[0]][neig[1]]

return summation\_xi\_xj

def energy(xi, neighbours):

summation\_xi\_xj = beta \* summation(xi, neighbours)

eta\_xi\_yi = eta \* xi \* Y[row][col]

h\_Xi = h \* xi

return h\_Xi - summation\_xi\_xj - eta\_xi\_yi

def error(X,Y1):

mis\_match = 0

for row in range(rows):

for col in range(cols):

if(X[row][col] != Y1[row][col]):

mis\_match +=1

return mis\_match/(311\*420)

#clean image

Y1 = mpimg.imread('Bayes\_grayscale.png')

Y1[Y1 == 0] = -1

status = True

print('Similarity in given noise image with origional image: '+str((1 - error(Y, Y1))\*100)+' %')

iteration = 0

while(status):

count = 0

for row in range(rows):

for col in range(cols):

xi = X[row][col]

neighbours = neighbors(row, col)

energy\_ = energy(xi, neighbours)

xi\_flip = xi \* -1

energy\_flip = energy(xi\_flip, neighbours)

if(energy\_flip < energy\_):

X[row][col] = xi\_flip

count += 1

if(count == 0):

status = False

iteration += 1

print("No. of pixel fliped in iteration "+str(iteration)+' is '+str(count))

print('Similarity now with origional image: '+str((1 - error(X, Y1))\*100)+' %')

imgplot = plt.imshow(X)

plt.savefig('cleaned.png')