```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
os.chdir("/content/drive/MyDrive/Colab Notebooks/STAT 431")
```

```
import math
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
from scipy import ndimage, misc
from PIL import Image
```

```
def add_gaussian_noise(img, sigma=5):
    clone = deepcopy(img)
    noise = np.abs(np.random.normal(0, scale=sigma, size=img.shape))

    subtraction = np.asarray(clone - noise, dtype=np.uint8)
    addition = np.asarray(clone + noise, dtype=np.uint8)

    boolz = np.array(clone == 255)
    gaussian_noise = np.where(boolz, subtraction, addition)

    plt.imshow(gaussian_noise, cmap=plt.get_cmap('gray'))
    plt.show()
    return gaussian_noise
```

```
import imageio

sigma = 20
for x in range(0,5):
  orig_img = imageio.imread("img"+str(x+1)+".png")
  orig_img.shape
  noisy_img = add_gaussian_noise(orig_img, sigma=sigma)
  imageio.imwrite("noisy_img_"+str(x)+".png", noisy_img)
```

```
# Function denoise_image() is the entry function. It calls another function get_posterior to get the estimated posterior probabilities
# p(Y=1|Y_neighbor). Setting the threshold to 0.5, we can get the restored image array Y from the posterior.
# Next, we strip the edges of the image array and return it. The following load_image function explains why we need to strip the array.

def denoise_image(filename, burn_in_steps, total_samples, logfile):
    posterior = get_posterior(filename, burn_in_steps, total_samples, logfile=logfile)
    denoised = np.zeros(posterior.shape, dtype=np.float64)
    denoised[posterior > 0.5] = 1
    return denoised[1:-1, 1:-1]
```

```
# In get_image(), we first read the PNG image into a numpy array, then we convert the RGB image to grayscale and re-scale the pixels to {-1,
# Here we add 0 paddings to the edges to help take care of corner cases when we search for each pixel’s neighbor later.
def get_image(filename):
    my_img = plt.imread(filename)
    img_gray = np.dot(my_img[..., :3], [0.2989, 0.5870, 0.1140])
    img_gray = np.where(img_gray > 0.5, 1, -1)
    img_padded = np.zeros([img_gray.shape[0] + 2, img_gray.shape[1] + 2])
    img_padded[1:-1, 1:-1] = img_gray
    return img_gray
```

```
for i in range(0,5):
  X = get_image("/content/drive/MyDrive/Colab Notebooks/STAT 431/noisy_img_"+str(i)+".png")
  plt.imshow(X)
  plt.show()
```

```
def sample(i, j, Y, X):
    markov_blanket = [Y[i - 1, j], Y[i, j - 1], Y[i, j + 1], Y[i + 1, j], X[i, j]]
    w = ITA * markov_blanket[-1] + BETA * sum(markov_blanket[:4])
    prob = 1 / (1 + math.exp(-2*w))
    return (np.random.rand() < prob) * 2 - 1
```

```
def get_posterior(filename, burn_in_steps, total_samples, logfile):
    X = get_image(filename)
    posterior = np.zeros(X.shape)
    print(X.shape)
    Y = np.random.choice([1, -1], size=X.shape)
    energy_list = list()
    for step in range(burn_in_steps + total_samples):
        for i in range(1, Y.shape[0]-1):
            for j in range(1, Y.shape[1]-1):
                y = sample(i, j, Y, X)
                Y[i, j] = y
                if y == 1 and step >= burn_in_steps:
                    posterior[i, j] += 1
        energy = -np.sum(np.multiply(Y, X))*ITA-(np.sum(np.multiply(Y[:-1], Y[1:]))+np.sum(np.multiply(Y[:, :-1], Y[:, 1:])))*BETA
        if step < burn_in_steps:
            energy_list.append(str(step) + "\t" + str(energy) + "\tB")
        else:
            energy_list.append(str(step) + "\t" + str(energy) + "\tS")
    posterior = posterior / total_samples

    file = open(logfile, 'w')
    for element in energy_list:
        file.writelines(element)
        file.write('\n')
    file.close()
    return posterior


def save_energy_plot(filename):
    x = np.genfromtxt(filename, dtype=None, encoding='utf8')
    its, energies, phases = zip(*x)
    its = np.asarray(its)
    energies = np.asarray(energies)
    phases = np.asarray(phases)
    burn_mask = (phases == 'B')
    samp_mask = (phases == 'S')
    assert np.sum(burn_mask) + np.sum(samp_mask) == len(x), 'Found bad phase'
    its_burn, energies_burn = its[burn_mask], energies[burn_mask]
    its_samp, energies_samp = its[samp_mask], energies[samp_mask]
    p1, = plt.plot(its_burn, energies_burn, 'r')
    p2, = plt.plot(its_samp, energies_samp, 'b')
    plt.title("energy")
    plt.xlabel('iteration number')
    plt.ylabel('energy')
    plt.legend([p1, p2], ['burn in', 'sampling'])
    plt.savefig('%s.png' % filename)
    plt.close()


def save_denoised_image(denoised_image , i):
    plt.imshow(denoised_image , cmap = 'gray')
    plt.title("denoised image")
    plt.savefig('Output/denoise_img_'+ str(i)+".png")
    plt.close()


if __name__ == '__main__':
  ITA = 1
  BETA = 1
  total_samples = 1000
  burn_in_steps = 10
  logfile = "Output/log_energy"
  for i in range(0,5):
    denoised_img = denoise_image("noisy_img_"+str(i)+".png", burn_in_steps=burn_in_steps,
                                 total_samples=total_samples, logfile=logfile)
    save_energy_plot(logfile)
    save_denoised_image(denoised_img , i)
```

✓ 1m 5s completed at 4:25 PM