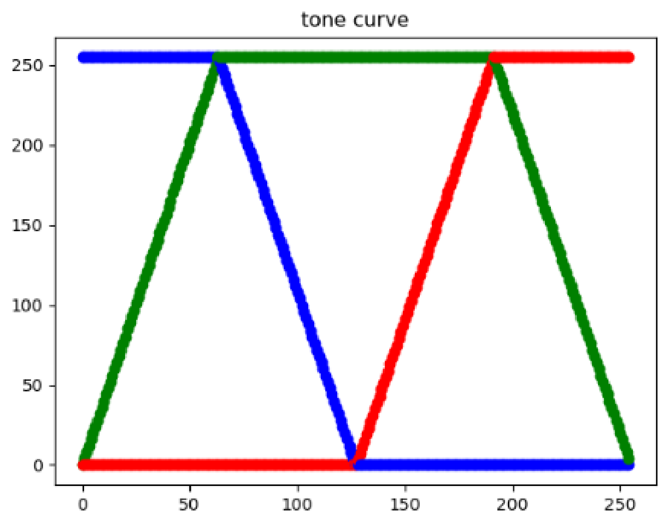
PS #2 Report

Problem 1: Pseudocoloring

Introduction

Pseudocoloring is a common technique used to increase the readability of grayscale images. The general method is to apply tone curves to each RGB channel. An example of these curves is shown below.



The position of the breaks in the piecewise functions are scaled to match the range of intensities in the grayscale image being colored.

Methods

The full implementation starts with finding the range of intensities in the grayscale image. To do this, `np.min()` and `np.max()` are used due to their speed over a simple double iterator.

```python
# Find minimum and maximum intensities in the grayscale image
i_min = np.min(img)
i_max = np.max(img)
```

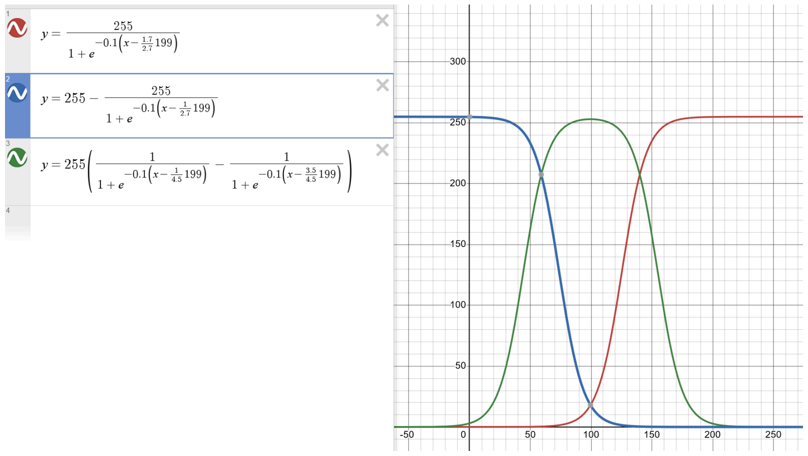
Once these values are found, the tone curves for each color channel can be defined. The implementation shown in class used piecewise functions for each color channel, however I chose to attempt an approach that removed the need to check which range of intensity values each one fell into and come up with a solution that was the same for every intensity value. The answer I settled on was logistic functions. Logistic functions are functions of the following form:

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

where L is the maximum value, k is a parameter that dictates the slope of the transition, and x_0 is the position of the midpoint. By massaging functions of this form for each color channel to match the shape we want, we can generate tone curves similar in shape to the linear piecewise functions that are defined the same for all intensities. After some tuning, I defined the following parameters for my tone curves.

```python
# Set parameters for each channel's tone curve
lut = []
k_br = 0.5
k_g = 0.1
span = i_max - i_min
br_mid = 1/2.3
g_mid = 1/4
```

`br_mid` and `g_mid` are fractions representing how far into the range of intensity values to place the midpoint of the logistic curve. Below is an example of the tone curves for an image with an intensity range of 199.

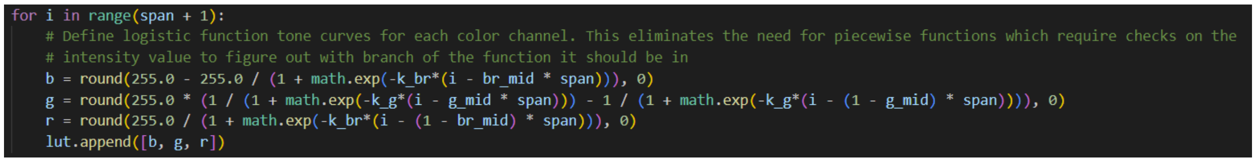


$$y = \frac{255}{1 + e^{-0.1\left(x - \frac{1.7}{2.7}199\right)}}$$

$$y = 255 - \frac{255}{1 + e^{-0.1\left(x - \frac{1}{2.7}199\right)}}$$

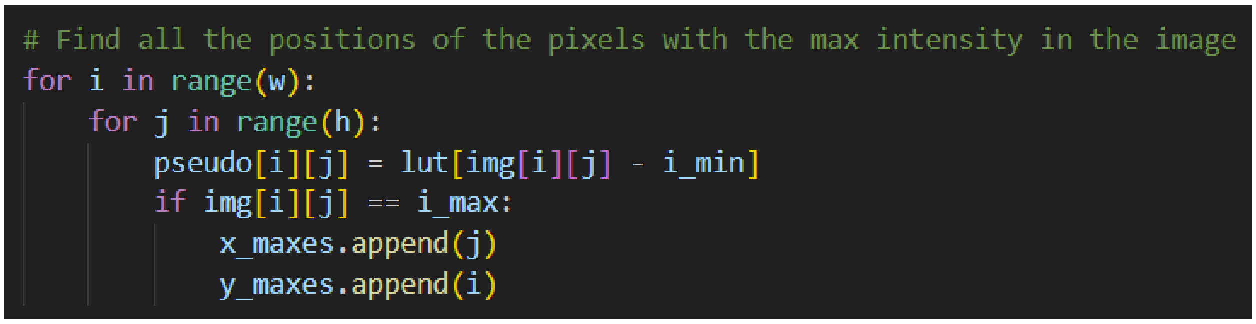
$$y = 255\left(\frac{1}{1 + e^{-0.1\left(x - \frac{1}{4.5}199\right)}} - \frac{1}{1 + e^{-0.1\left(x - \frac{3.5}{4.5}199\right)}}\right)$$

The curves are implemented in Python as follows and used to generate a lookup array with [B, G, R] values for every intensity value in the original image.



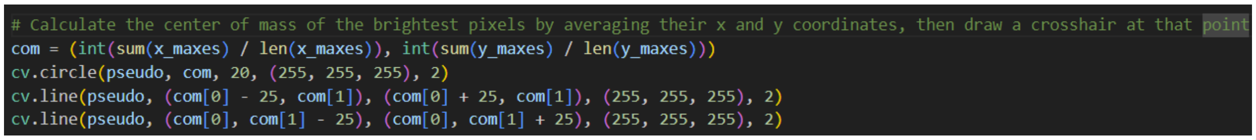
```python
for i in range(span + 1):
    # Define logistic function tone curves for each color channel. This eliminates the need for piecewise functions which require checks on the
    # intensity value to figure out with branch of the function it should be in
    b = round(255.0 - 255.0 / (1 + math.exp(-k_br*(i - br_mid * span))), 0)
    g = round(255.0 * (1 / (1 + math.exp(-k_g*(i - g_mid * span))) - 1 / (1 + math.exp(-k_g*(i - (1 - g_mid) * span)))), 0)
    r = round(255.0 / (1 + math.exp(-k_br*(i - (1 - br_mid) * span))), 0)
    lut.append([b, g, r])
```

Once the LUT is generated, the original image is iterated through again. Based on the intensity of each pixel, a BGR value from the LUT is selected and applied to a new image at the same position. At the same time, the positions of the pixels with maximum intensity are tracked.



```python
# Find all the positions of the pixels with the max intensity in the image
for i in range(w):
    for j in range(h):
        pseudo[i][j] = lut[img[i][j] - i_min]
        if img[i][j] == i_max:
            x_maxes.append(j)
            y_maxes.append(i)
```

Finally, once the new color image is generated, the x and y values of the maximum intensity pixels are averaged to find the centroid and a crosshair is drawn there.



```python
# Calculate the center of mass of the brightest pixels by averaging their x and y coordinates, then draw a crosshair at that point
com = (int(sum(x_maxes) / len(x_maxes)), int(sum(y_maxes) / len(y_maxes)))
cv.circle(pseudo, com, 20, (255, 255, 255), 2)
cv.line(pseudo, (com[0] - 25, com[1]), (com[0] + 25, com[1]), (255, 255, 255), 2)
cv.line(pseudo, (com[0], com[1] - 25), (com[0], com[1] + 25), (255, 255, 255), 2)
```

Results

The results of the pseudocoloring for each of the five given images are below.