Lab 4: Pointwise Operations and Gamma

Course Title: Image Processing I (Spring 2022)

Course Number: ECE 63700

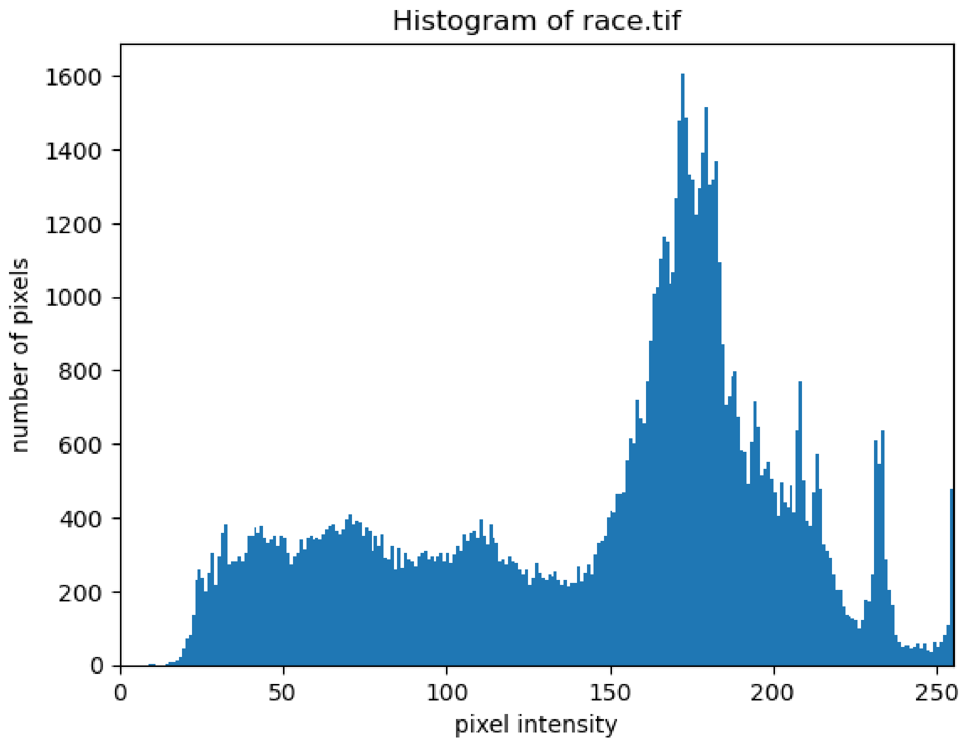
Instructor: Prof. Charles A. Bouman

Author: **Zhankun Luo**

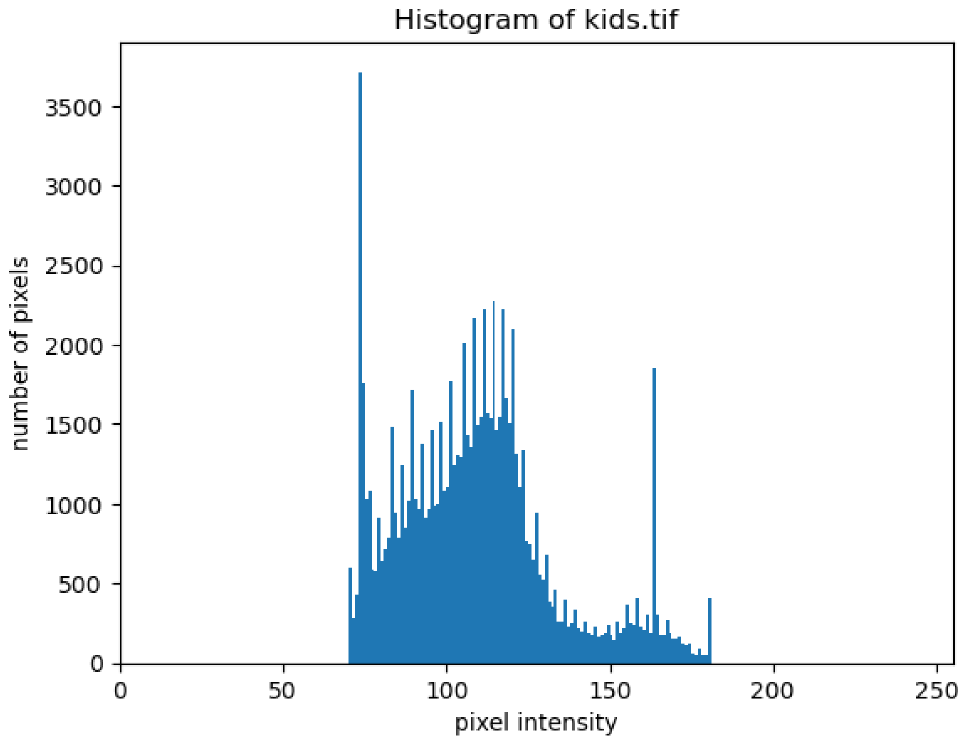
1. Histogram of an Image

1.1. the histogram of `race.tif` and `kids.tif`

solution



Histogram of Gray Scale Image **race.tif**



Histogram of Gray Scale Image **kids.tif**

2. Histogram Equalization

2.1. the function `equalize()`

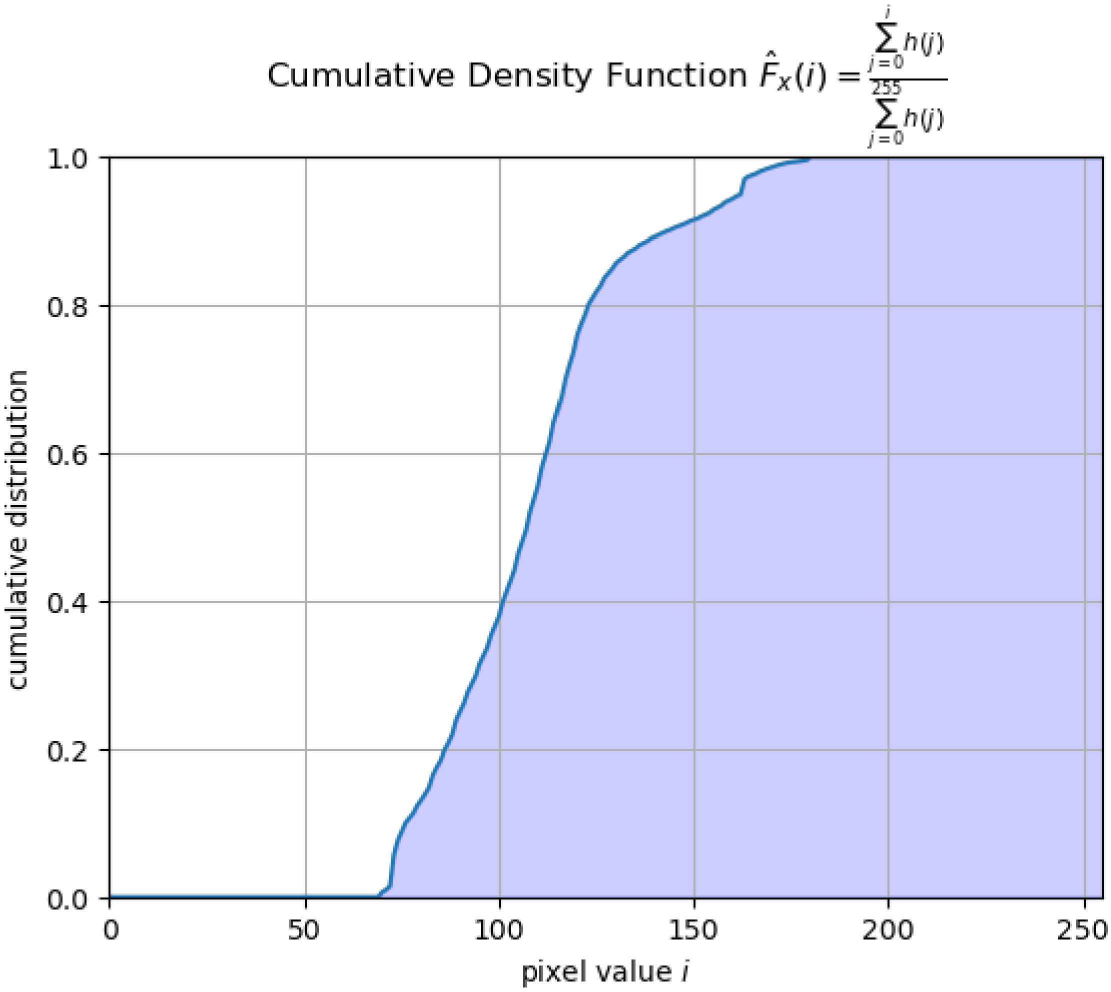
solution

```
MAX = 255
def _hist(x: ndarray) -> ndarray:
    hist = zeros((MAX+1,)).astype(int)
    for i in x:
        for j in i:
            hist[j] += 1
    return hist

def equalize(x: ndarray) -> ndarray:
    H, W = x.shape
    cdf_ref = _hist(x).cumsum() / (H * W)
    x_equ = zeros((H, W)).astype(np.float)
    for i in range(H):
        for j in range(W):
            x_equ[i][j] = cdf_ref[ x[i][j] ]
    max_ref, min_ref = x_equ.max(), x_equ.min()
    x_equ = MAX * (x_equ-min_ref)/(max_ref-min_ref)
    return x_equ.astype(np.uint8)
```

2.2. the labeled plot of CDF $\hat{F}_x(i)$ for `kids.tif`

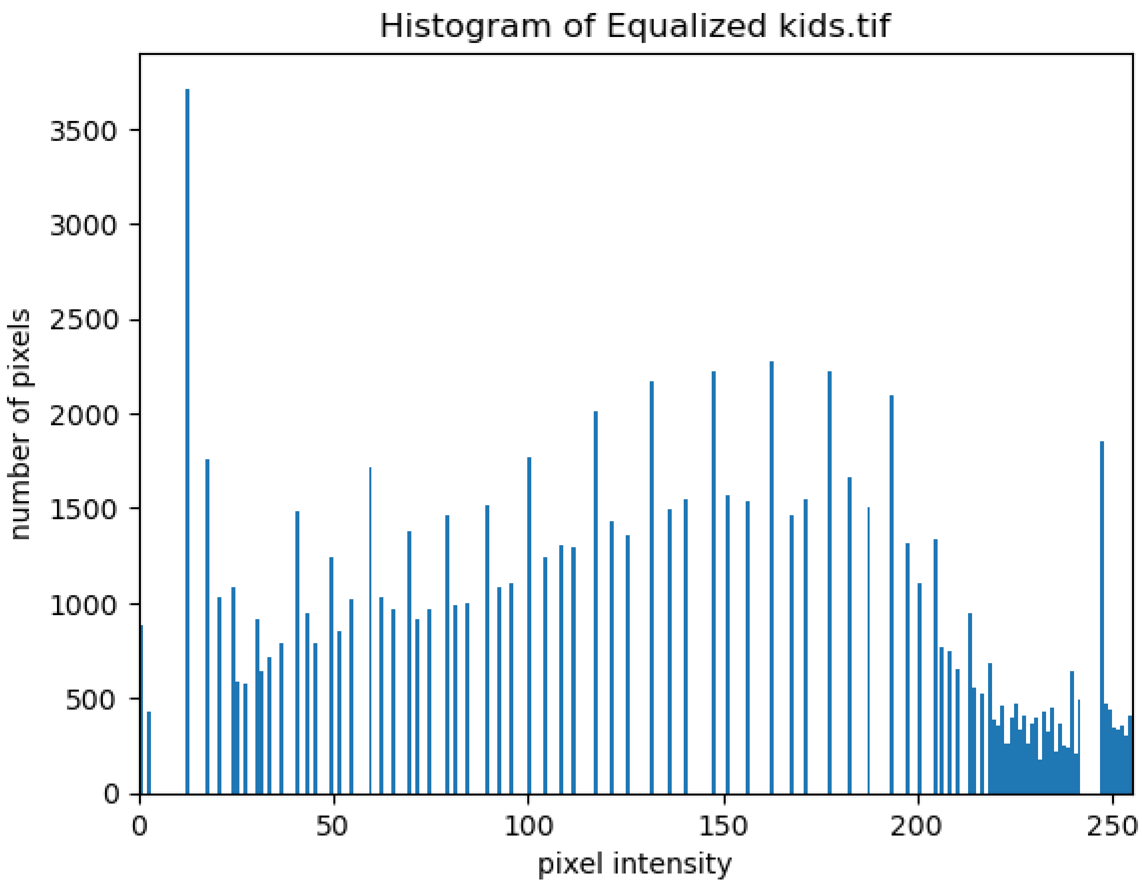
solution



Cumulative Distribution Function for Gray Scale Image **kids.tif**

2.3. the histogram of the equalized image for `kids.tif`

solution



Histogram of the Equalized Image for Gray Scale Image **kids.tif**

2.4. the equalized image for `kids.tif`

solution

Equalized Image for Gray Scale Image **kids.tif**

3. Contrast Stretching

3.1. the function `stretch()`

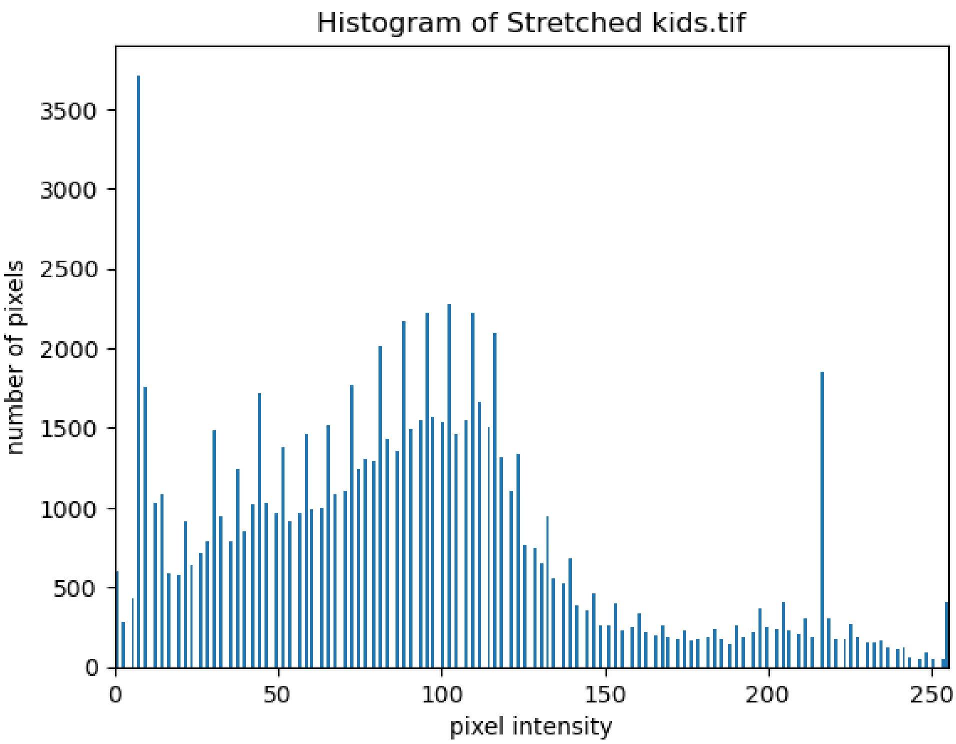
solution

```
MAX = 255
def stretch(x: ndarray, T1: np.uint8, T2: np.uint8) -> ndarray:
    H, W = x.shape
    correspond = [0] * (T1+1) + \
        [round(MAX*(e+1-T1)/(T2-T1)) for e in range(T1, T2)] \
        + [MAX] * (MAX-T2)
    x_stre = zeros((H, W)).astype(np.uint8)
    for i in range(H):
        for j in range(W):
            x_stre[i][j] = correspond[ x[i][j] ]
    return x_stre
```

3.2. the stretched image and its histogram for `kids.tif`

We set $T_1 = 70, T_2 = 180$ for `kids.tif`, where $70, 180$ are the minimal and maximum pixel value in `kids.tif`

Stretched Image for Gray Scale Image **kids.tif**

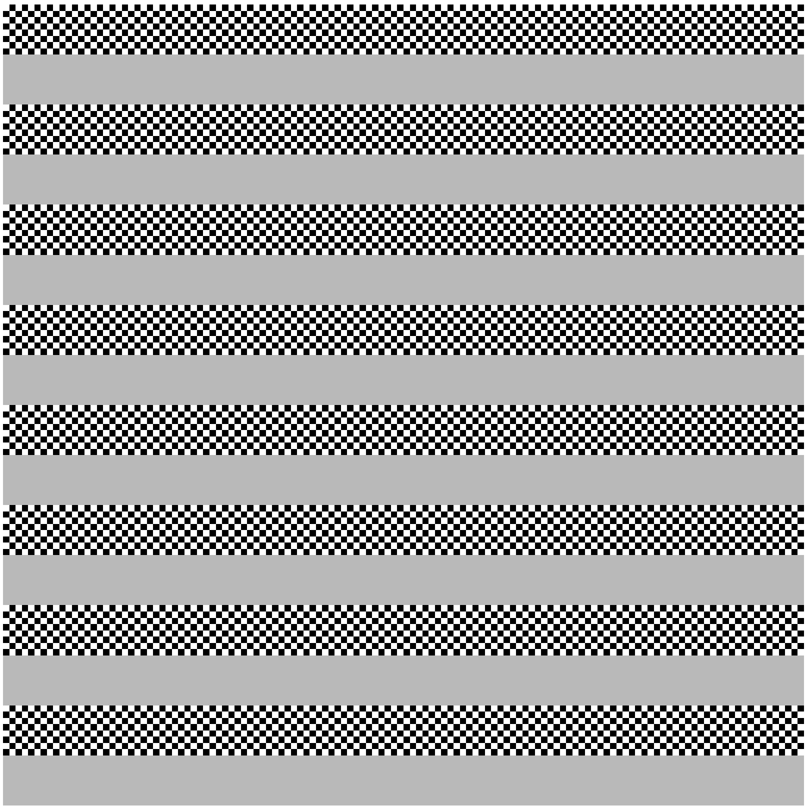


Histogram of Stretched Image for Gray Scale Image **kids.tif**

4. Gamma (γ)

4.2. Determining the Gamma of Your Computer Monitor

4.2.1. the image corresponding to the matching gray level g



256x256 Arrays whose Checkerboard Pattern Matches Constant Gray Level **g**

4.2.2. the expression that relates the matching gray level to γ

By letting $I_c = I_g$, where I_c is the perceived intensity of the checkerboard, I_g is the perceived intensity for gray level g

$$\left.\begin{aligned} I_c &= \tfrac{I_{255}}{2} \\ I_g &= I_{255} \cdot \left(\tfrac{g}{255}\right)^{\gamma} \end{aligned}\right\} \overset{I_c = I_g}{\Longrightarrow} \frac{1}{2} = \left(\frac{g}{255}\right)^{\gamma}$$

Thus, we can estimate γ based on the matched gray level g

$$\gamma = \frac{\log\left(\frac{1}{2}\right)}{\log g - \log 255} = \frac{\log 2}{\log 255 - \log g}$$

4.2.3. the measured gray level g and γ

We adjust the value of gray level g until our checkerboard pattern matches constant gray level, and the matched gray level $g = 185$ for my monitor.

The corresponding estimated γ for the match gray level $g = 185$ is

$$\gamma = \frac{\log 2}{\log 255 - \log g} = \frac{\log 2}{\log 255 - \log 185} \approx 2.1600$$

4.3. Gamma Correction

4.3.1. the corrected image for `linear.tif` and itself

linear.tif*(left) and its Corrected Image with Gamma=2.1600(right)*

4.3.2. the formula ψ_γ used to correct `linear.tif`

Suppose we apply the transform ψ_γ on the gray level g_0 of the original image `linear.tif` to obtain the gray level g of corrected image

$$g := \psi_\gamma(g_0)$$

We know that the mapping φ_γ from the gray level g of corrected image to the perceived intensity I_g is given by

$$I_g := \varphi_\gamma(g) = I_{255} \cdot \left(\frac{g}{255}\right)^\gamma$$

Let perceived intensity I_g be proportional to gray level g_0 of original image, where $k > 0$

$$I_g := \varphi_\gamma\left(\psi_\gamma(g_0)\right) = k \cdot g_0$$

We conclude the relationship between our transform ψ_γ and the mapping of monitor φ_γ , where id is an identity function

$$\varphi_\gamma \circ \psi_\gamma = k \circ \mathrm{id} \Longrightarrow \psi_\gamma = \varphi_\gamma^{-1} \circ k \circ \mathrm{id}$$

So, the the transform ψ_γ on the gray level g_0 of the original image is

$$\psi_\gamma(g_0) = \varphi_\gamma^{-1}(k \cdot g_0) = 255 \cdot \left(k \cdot \frac{g_0}{I_{255}}\right)^{\frac{1}{\gamma}}$$

We can choose k to ensure $\varphi_\gamma(\psi_\gamma(255)) = I_{255}$

$$\varphi_\gamma(\psi_\gamma(255)) = I_{255} \Longrightarrow k := \left(\frac{I_{255}}{255}\right)$$

Final formula used to correct `linear.tif` is, where $\gamma = 2.1600$ for our monitor

$$\psi_\gamma : g_0 \mapsto 255 \cdot \left(\frac{g_0}{255}\right)^{\frac{1}{\gamma}}$$

4.3.3. the corrected image for `gamma15.tif` and itself

gamma15.tif*(left) and its Corrected Image with Gamma=2.1600/1.5=1.4400(right)*

4.3.4. the procedure ψ used to change the gamma correction of the original image

Consider the gray pixel g_0 of original image, the gray pixel g' of corrected image with initial $\gamma_0 = 1.5$, ψ_{γ_0}, the gray pixel g of changed image after applying our procedure ψ, and the perceived intensity I_g

$$\psi_{\gamma_0} : g_0 \longmapsto g' := 255 \cdot \left(\frac{g_0}{255}\right)^{\frac{1}{\gamma_0}}$$
$$\psi : g' \longmapsto g := \psi(g')$$
$$\varphi_\gamma : g \longmapsto I_g := I_{255} \cdot \left(\frac{g}{255}\right)^{\gamma}$$

Let perceived intensity I_g be proportional to gray level g_0 of original image, $k = \frac{I_{255}}{255} > 0$

$$I_g := \varphi_\gamma\left(\psi(\psi_{\gamma_0}(g_0))\right) = k \cdot g_0$$

We conclude the relationship between our procedure ψ_γ and the mapping of monitor φ_γ , the initial Gamma correction ψ_{γ_0}, where id is an identity function

$$\varphi_\gamma \circ \psi \circ \psi_{\gamma_0} = k \circ \text{id} \Longrightarrow \psi = \varphi_\gamma^{-1} \circ k \circ \psi_{\gamma_0}^{-1}$$

notice in the previous section $\varphi_\gamma^{-1} \equiv \psi_\gamma \circ k^{-1} \circ \text{id}$

$$\psi = (\psi_\gamma \circ k^{-1} \circ \text{id}) \circ k \circ \psi_{\gamma_0}^{-1} = \psi_\gamma \circ \psi_{\gamma_0}^{-1}$$

notice the properties of Gamma correction $\psi_\gamma, \gamma > 0$

$$\psi_{\gamma_0^{-1}} = \psi_{\gamma_0}^{-1}, \quad \psi_{\gamma_1} \circ \psi_{\gamma_2} = \psi_{\gamma_1 \gamma_2}, \quad \forall \gamma_0, \gamma_1, \gamma_2 > 0$$

The procedure ψ used to change the gamma correction of the original image is

$$\psi = \psi_\gamma \circ \psi_{\gamma_0}^{-1} = \psi_{\frac{\gamma}{\gamma_0}}$$

$$\psi = \psi_{\frac{\gamma}{\gamma_0}} : g' \mapsto 255 \cdot \left(\frac{g'}{255}\right)^{\frac{\gamma_0}{\gamma}}$$

where $\gamma_0 = 1.5$, the Gamma for our monitor $\gamma = 2.1600$ and $\frac{\gamma}{\gamma_0} = 1.4400$

Appendix

Python codes for functions

utils.py

```
import matplotlib.pyplot as plt
from matplotlib import rcParams
from numpy import ndarray, zeros, ones, array, log, vectorize
from numpy.matlib import repmat
import numpy as np
MAX = 255

# section 1
def histogram(x: ndarray) -> None:
    plt.hist(x.flatten(), bins=list(range(MAX+1)))
    plt.xlim([0, MAX])
    plt.xlabel("pixel intensity")
    plt.ylabel("number of pixels")

def _hist(x: ndarray) -> ndarray:
    hist = zeros((MAX+1,)).astype(int)
    for i in x:
        for j in i:
            hist[j] += 1
    return hist

# section 2
def _match_hist(cdf: list, cdf_ref: list) -> list:
    correspond = [len(cdf)-1]
    for i in range(len(cdf)-2, -1, -1):
        j = correspond[0]
        while True:
            if (j <= 0) or (cdf[j] <= cdf_ref[i]):
                correspond.insert(0, j)
                break
            j -= 1
    return correspond

"""Zhankun's method with less computation"""
```

```python
def equalize_quick(x: ndarray) -> ndarray:
    H, W = x.shape
    cdf_ref = _hist(x).cumsum() / (H * W)
    cdf = [(i+1)/(MAX+1) for i in range(MAX)]
    correspond = _match_hist(cdf, cdf_ref)
    x_equ = zeros((H, W)).astype(np.uint8)
    for i in range(H):
        for j in range(W):
            x_equ[i][j] = correspond[ x[i][j] ]
    return x_equ

"""the method in lab 4 instruction PDF"""
def equalize(x: ndarray) -> ndarray:
    H, W = x.shape
    cdf_ref = _hist(x).cumsum() / (H * W)
    x_equ = zeros((H, W)).astype(np.float)
    for i in range(H):
        for j in range(W):
            x_equ[i][j] = cdf_ref[ x[i][j] ]
    max_ref, min_ref = x_equ.max(), x_equ.min()
    x_equ = MAX * (x_equ-min_ref)/(max_ref-min_ref)
    return x_equ.astype(np.uint8)

def plot_CDF(x: ndarray) -> None:
    H, W = x.shape
    cdf_ref = _hist(x).cumsum() / (H * W)
    plt.fill_between(range(MAX+1), cdf_ref, color = 'b', alpha=0.2)
    plt.plot(cdf_ref)
    plt.xlim([0, MAX])
    plt.ylim([0, 1])
    plt.grid()
    plt.xlabel(r"pixel value $i$")
    plt.ylabel("cumulative distribution")
    plt.title("Cumulative Density Function " +
    r"$\hat{F}_{x}(i)=\frac{\sum_{j=0}^{i} h(j)}{\sum_{j=0}^{255}
h(j)}$")

# section 3
def stretch(x: ndarray, T1: np.uint8, T2: np.uint8) -> ndarray:
    H, W = x.shape
    correspond = [0] * (T1+1) + \
        [round(MAX*(e+1-T1)/(T2-T1)) for e in range(T1, T2)] \
        + [MAX] * (MAX-T2)
```

```
    x_stre = zeros((H, W)).astype(np.uint8)
    for i in range(H):
        for j in range(W):
            x_stre[i][j] = correspond[ x[i][j] ]
    return x_stre

# section 4
def generate_pattern(g: np.uint8):
    num_strip = 8
    num_unit_row = 4
    num_unit_col = 64
    H, W = num_strip*2*(num_unit_row*4), num_unit_col*4
    x = g * ones((H, W)).astype(np.uint8)
    unit =  array([
        [MAX, MAX, 0, 0],
        [MAX, MAX, 0, 0],
        [0, 0, MAX, MAX],
        [0, 0, MAX, MAX]])
    stripe = repmat(unit, num_unit_row, num_unit_col)
    height_stripe = stripe.shape[0]
    for i in range(0, H, 2*height_stripe):
        x[i:i+height_stripe] = stripe
    return x

def compute_gamma(g: np.uint8) -> float:
    return log(0.5) / log(g/MAX)

def correct_gamma(x: ndarray, gamma: float) -> ndarray:
    func = lambda t: round(MAX*pow(t/MAX, 1./gamma))
    f = vectorize(func)
    return f(x).astype(np.uint8)
```

Python codes for solutions

solution to section 1: `soln_1.py`

```
import sys
from os.path import dirname, join
sys.path.insert(0, dirname(dirname(__file__)))
from PIL import Image
import matplotlib.pyplot as plt
from numpy import array
from src.utils import histogram

if __name__ == "__main__":
    root = "resource"
    for index, img in list(zip(['a', 'b'],["race.tif", "kids.tif"])):
        path_img = join(root, img)
        x = array(Image.open(path_img))
        histogram(x)
        plt.title("Histogram of {}".format(img))
        plt.savefig("result/fig_1{}.png"
                    .format(index), bbox_inches='tight')
        plt.show()
```

solution to section 2: `soln_2.py`

```
import sys
from os.path import dirname, join
sys.path.insert(0, dirname(dirname(__file__)))
from PIL import Image
import matplotlib.pyplot as plt
from numpy import array
from src.utils import histogram, equalize, plot_CDF



if __name__ == "__main__":
    img = "kids.tif"
    path_img = join("resource", img)
    x = array(Image.open(path_img))
    plot_CDF(x)
    plt.savefig("result/fig_2_2.png", bbox_inches='tight')
    plt.show()
    x_equ = equalize(x)
    histogram(x_equ)
    plt.title("Histogram of Equalized {}".format(img))
    plt.savefig("result/fig_2_3.png", bbox_inches='tight')
    plt.show()
    Image.fromarray(x_equ).save("result/fig_2_4.tif")
```

solution to section 3: `soln_3.py`

```
import sys
from os.path import dirname, join
sys.path.insert(0, dirname(dirname(__file__)))
from PIL import Image
import matplotlib.pyplot as plt
from numpy import array
from src.utils import histogram, stretch

if __name__ == "__main__":
    img = "kids.tif"
    path_img = join("resource", img)
    x = array(Image.open(path_img))
    T1, T2 = x.min(), x.max()
    print("T1: {}, T2: {}".format(T1, T2))
    x_stre = stretch(x, T1, T2)
    Image.fromarray(x_stre).save("result/fig_3_2a.tif")
    histogram(x_stre)
    plt.title("Histogram of Stretched {}".format(img))
    plt.savefig("result/fig_3_2b.png", bbox_inches='tight')
    plt.show()
```

solution to section 4: `soln_4.py`

```python
import sys
from os.path import dirname, join
sys.path.insert(0, dirname(dirname(__file__)))
from PIL import Image
import matplotlib.pyplot as plt
from numpy import array
from src.utils import compute_gamma, generate_pattern, correct_gamma

if __name__ == "__main__":
    g = 185
    pattern = generate_pattern(g)
    Image.fromarray(pattern).save("result/fig_4_2_1.tif")
    gamma = compute_gamma(g)
    print("gamma: {}".format(gamma))
    img0 = "linear.tif"
    path_img0 = join("resource", img0)
    x0 = array(Image.open(path_img0))
    x0_correct = correct_gamma(x0, gamma)
    Image.fromarray(x0_correct).save("result/fig_4_3_1.tif")
    img1 = "gamma15.tif"
    path_img1 = join("resource", img1)
    x1 = array(Image.open(path_img1))
    x1_correct = correct_gamma(x1, gamma/1.5)
    Image.fromarray(x1_correct).save("result/fig_4_3_2.tif")
```