**北京邮电大学软件学院**

**2022-2023学年第二学期实验报告**

**课程名称：** 多媒体技术与应用

**项目名称： 实验二 灰度变换与直方图均衡**

**项目完成人：**

**姓名： 王宇涵 学号： 2020211730**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_李朝晖\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2023年 3 月 30 日**

1. **实验目的**

⑴ 深刻理解直接灰度变换及直方图均衡原理；

⑵ 掌握直接灰度变换及直方图均衡技术；

1. **实验内容**
2. 在实验1的基础上，编制程序，实现图像文件的直接灰度变换功能，并能显示灰度变换后图像效果，展示变换前后直方图的变化。
3. 在上面实验的基础上，编制程序，实现图像文件的直方图均衡功能,并能显示直方图均衡前后直方图变化
4. **实验环境**

Windows10操作系统

Pycharm(编译器)

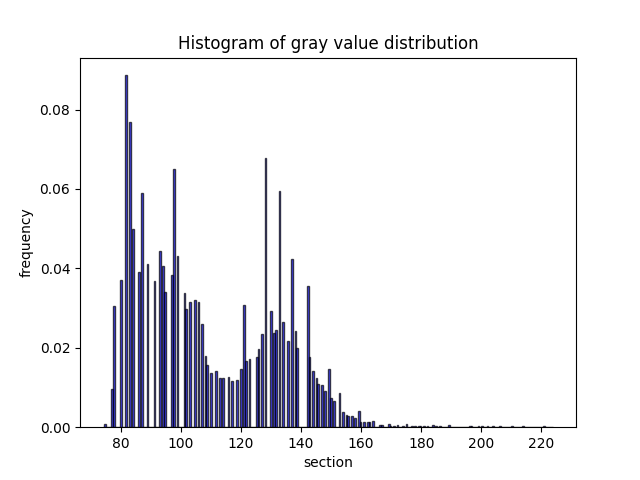
Python3.8

1. **实验结果**
2. **直接灰度变换**
   1. **线性灰度变换,图像反转**
      1. 图片1

原图:



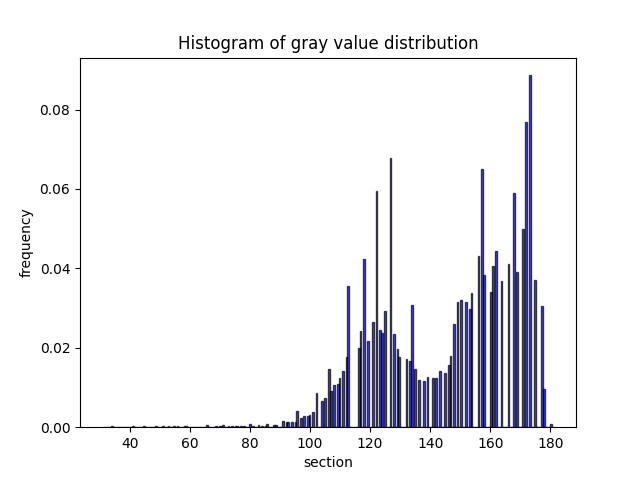
直方图:



反转后效果图



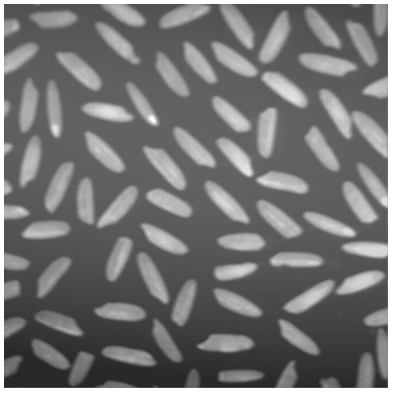
直方图:



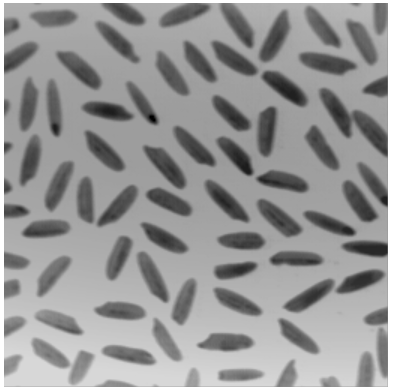
图片2和图片3仅展示原图和反转后效果, 因为反转相对简单, 不写得太冗余.

* + 1. 图片2

原图:



反转后:

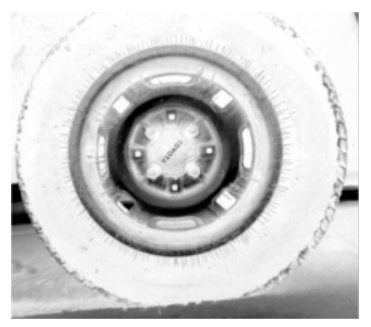


* + 1. 图片3

原图:



反转后图像:



* 1. **非线性灰度变换,对数变换**

对数变换将图像的低灰度值部分扩展，将其高灰度值部分压缩，以达到强调图像低灰度部分的目的；同时可以很好的压缩像素值变化较大的图像的动态范围，目的是突出我们需要的细节。反对数变换则与对数函数不同的是，强调的是图像的高灰度部分，对数变换公式如下：

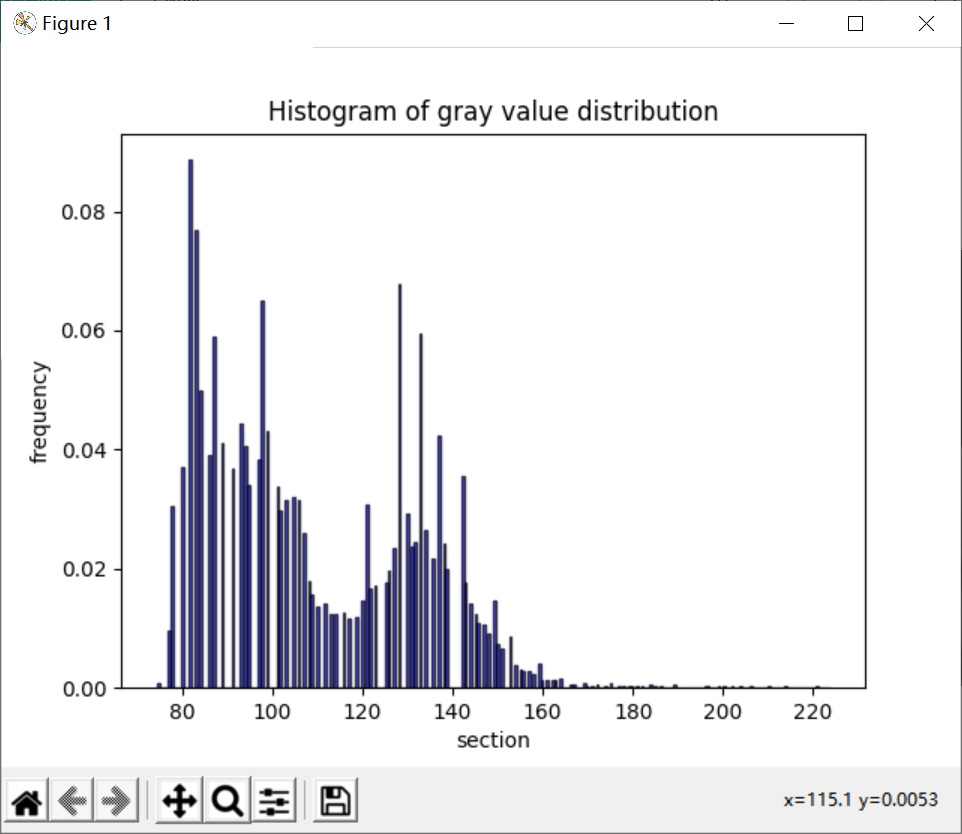
对数变换的通用公式是： s=c log(1+r);

* + 1. 图片1

原图:



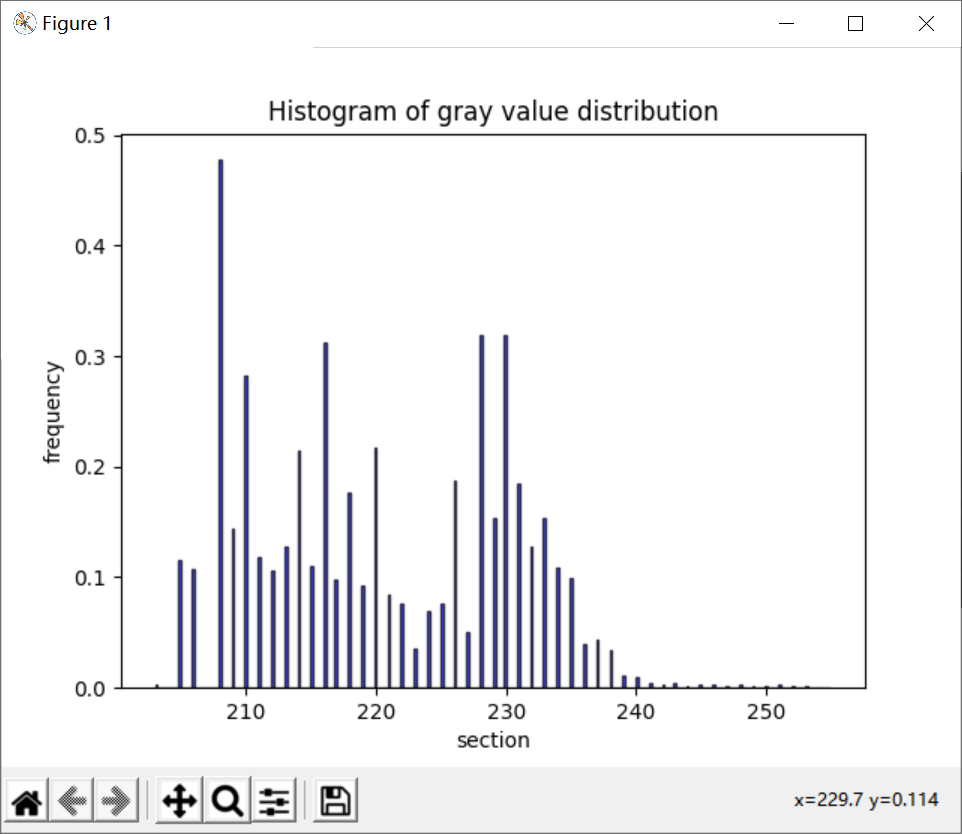
原图直方图:



变换后图片:

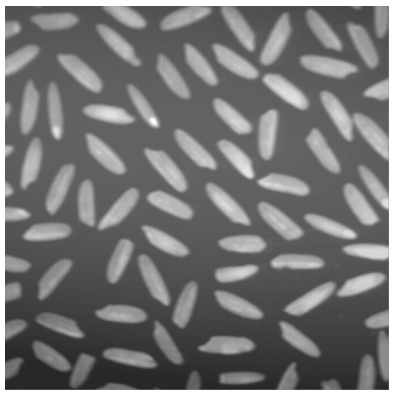


变换后直方图:

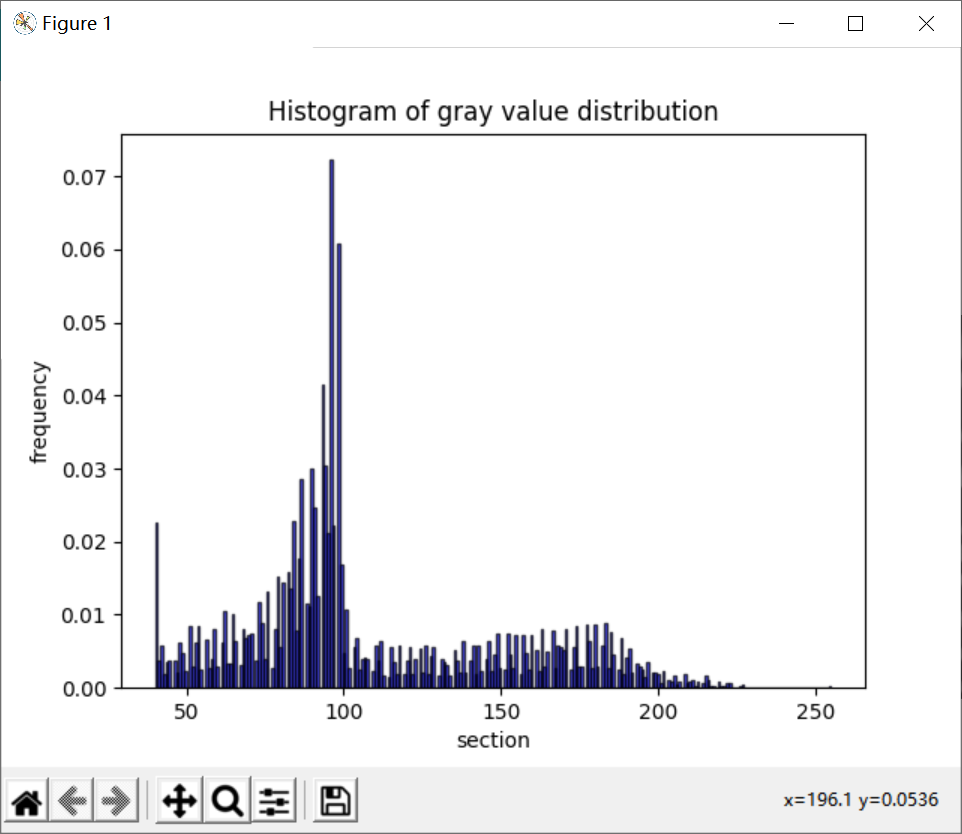


* + 1. 图片2

原图:



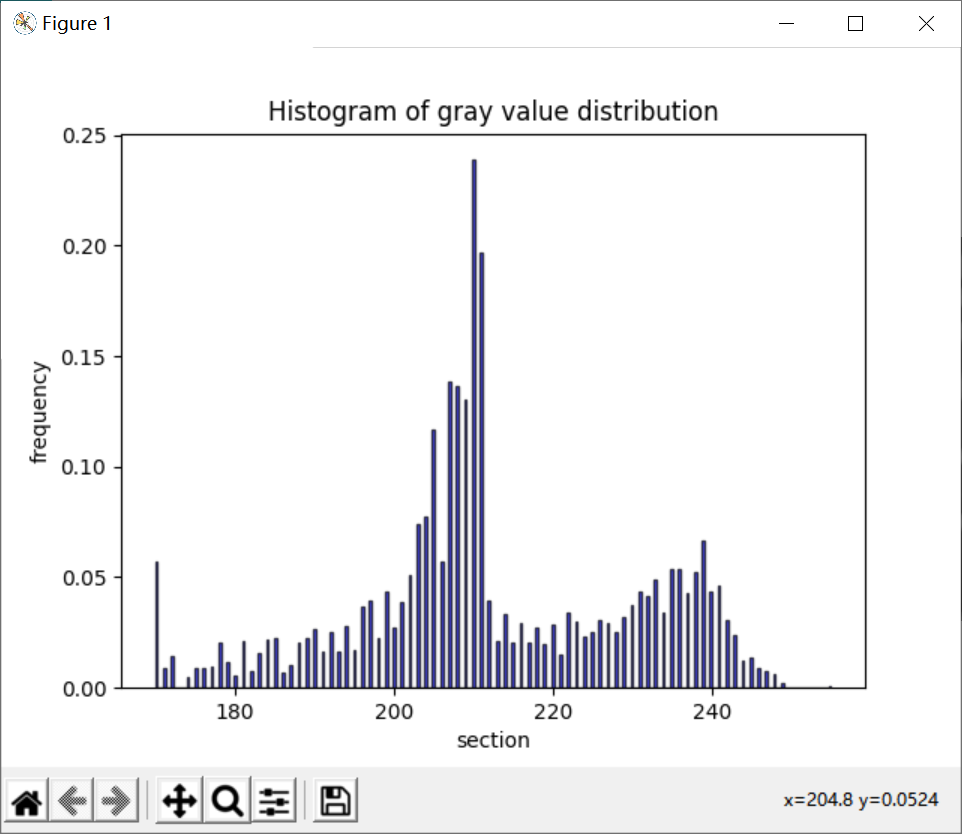
原图直方图:



变换后图象:



变换后直方图:

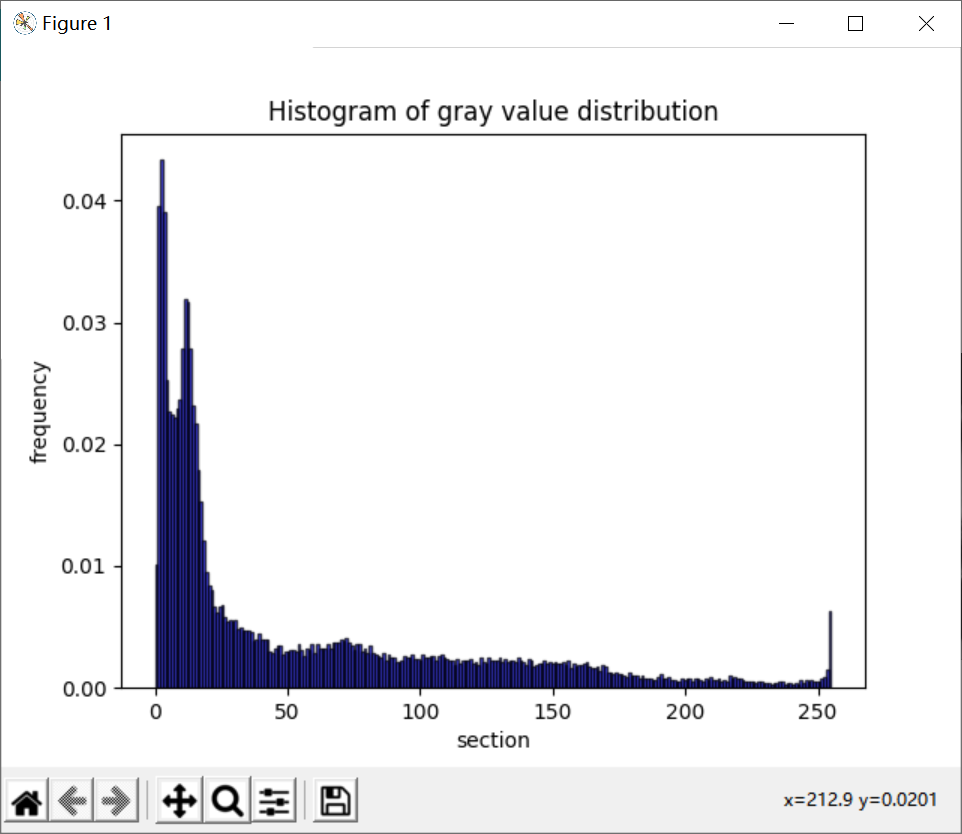


* + 1. 图片3

原图:



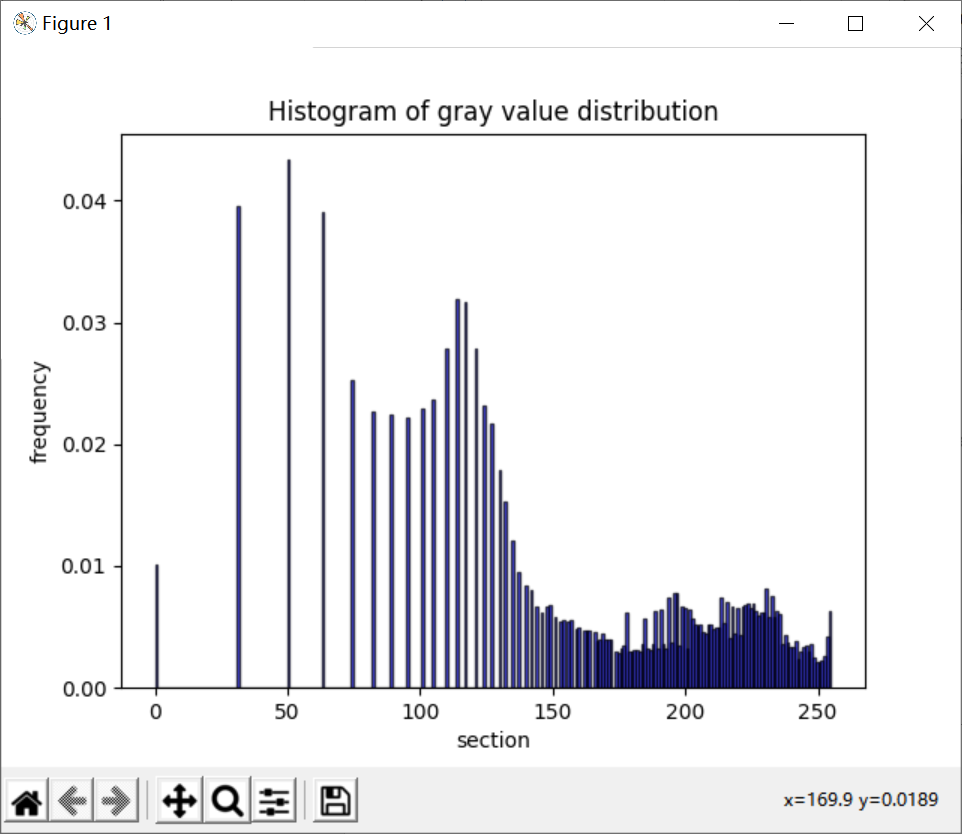
原图直方图:



变换后图象:



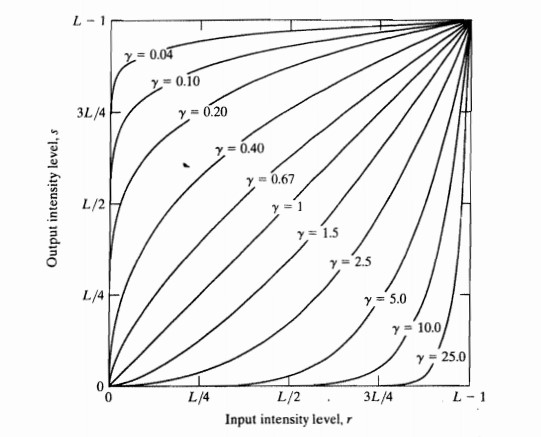
变换后直方图:



* 1. **非线性灰度变换,伽马变换**

伽马变换的公式为：s=crγ

其中c和γ为正常数。  
伽马变换的效果与对数变换有点类似，当γ>1时将较窄范围的低灰度值映射为较宽范围的灰度值，同时将较宽范围的高灰度值映射为较窄范围的灰度值；当γ<1时，情况相反，与反对数变换类似。其函数曲线如下：



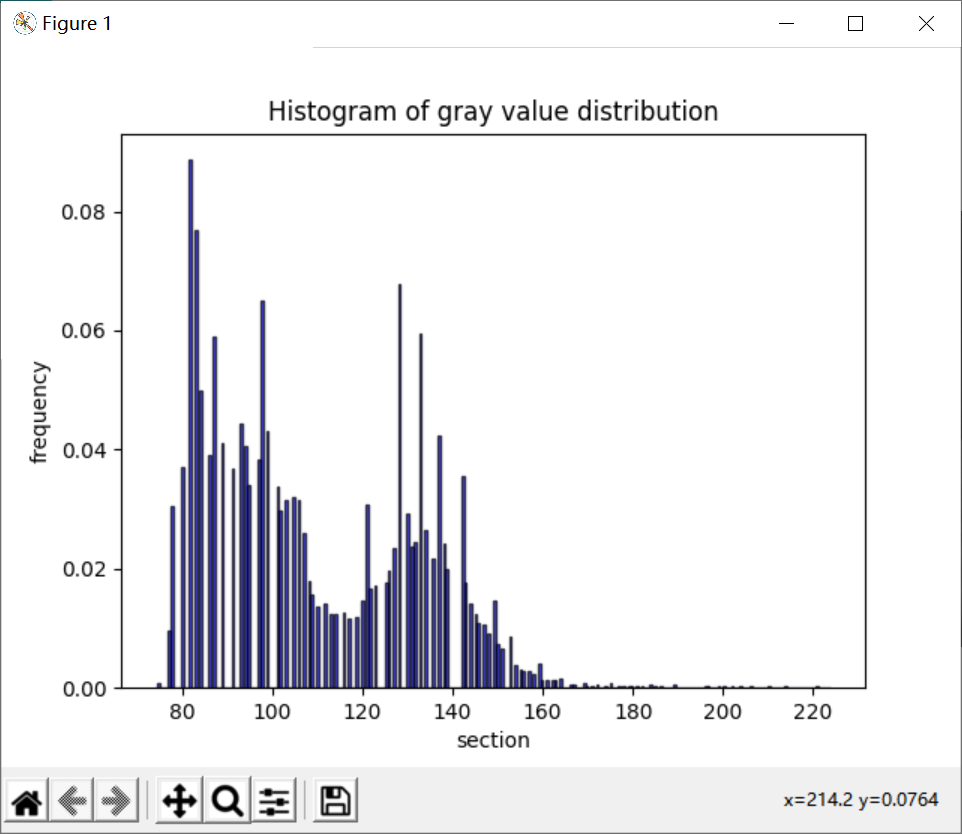
当γ<1时，γ的值越小，对图像低灰度值的扩展越明显；当γ>1时，γ的值越大，对图像高灰度值部分的扩展越明显。这样就能够显示更多的图像的低灰度或者高灰度细节。

* + 1. 图片1 (c = 0.75, γ = 1.18 )

原图:



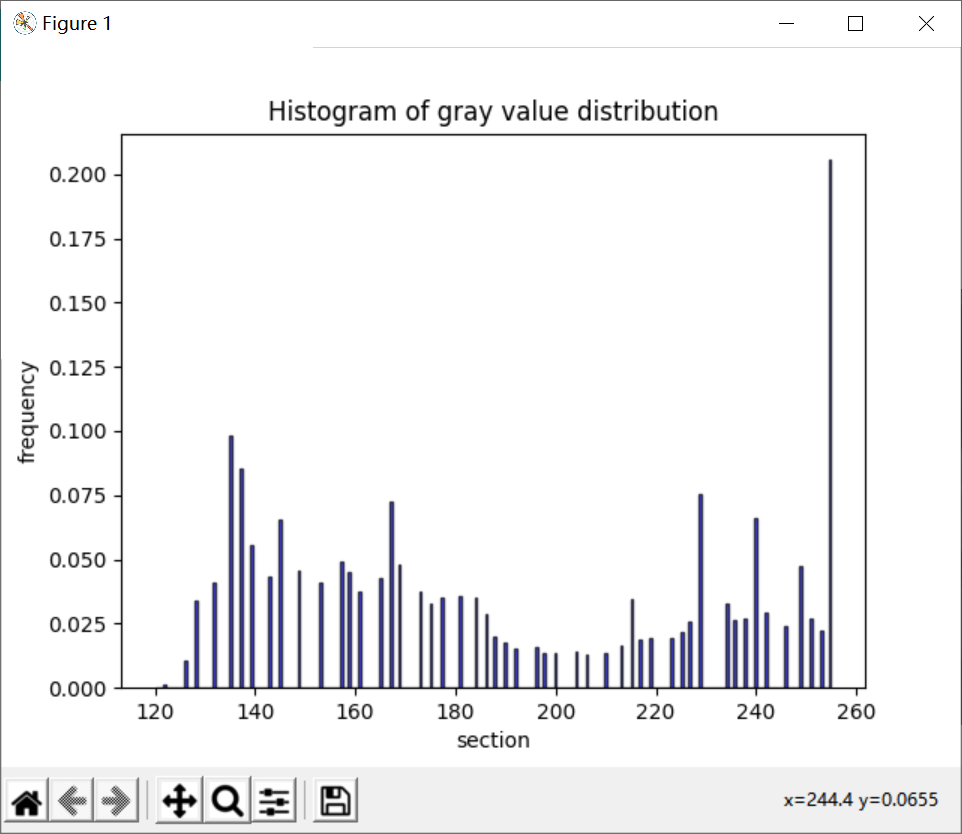
原图直方图:



变换后图象:

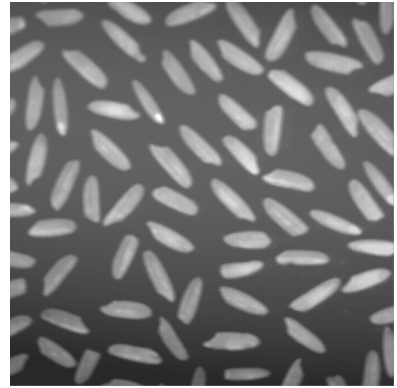


变换后直方图:

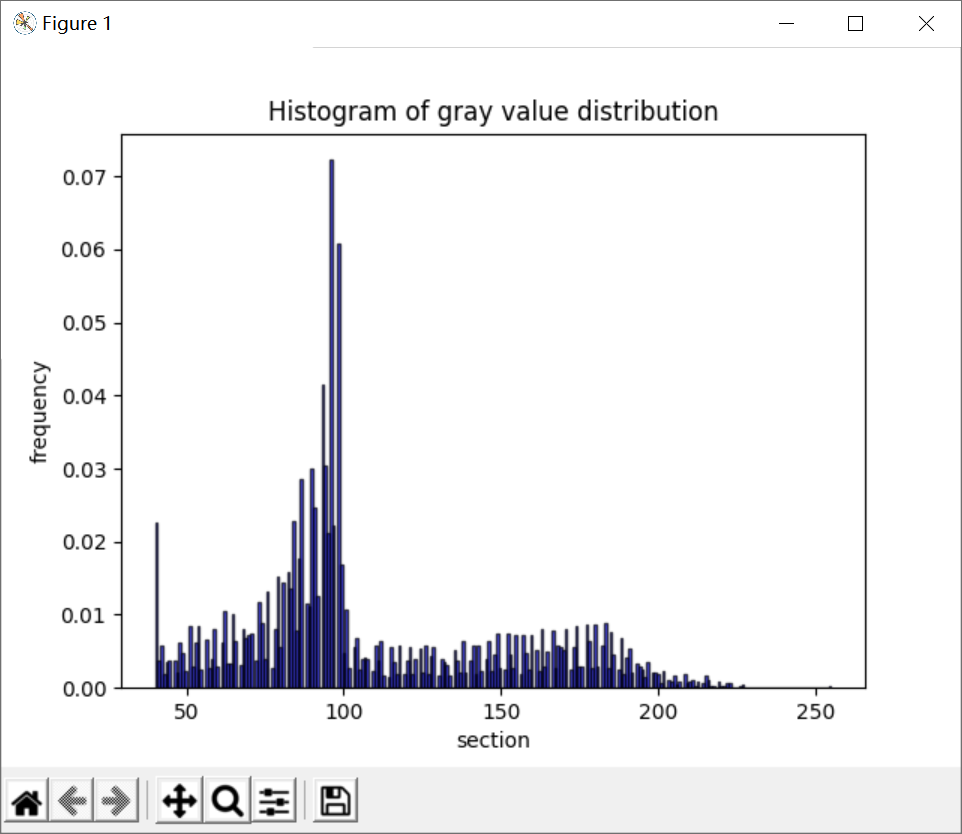


1. 图片2 (c = 0.75, γ = 1.13 )

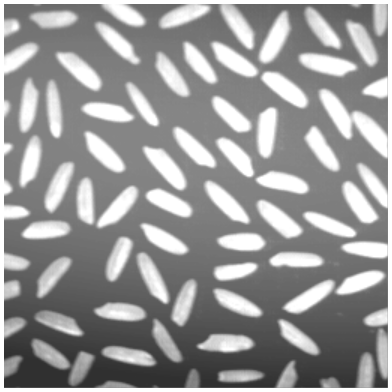
原图:



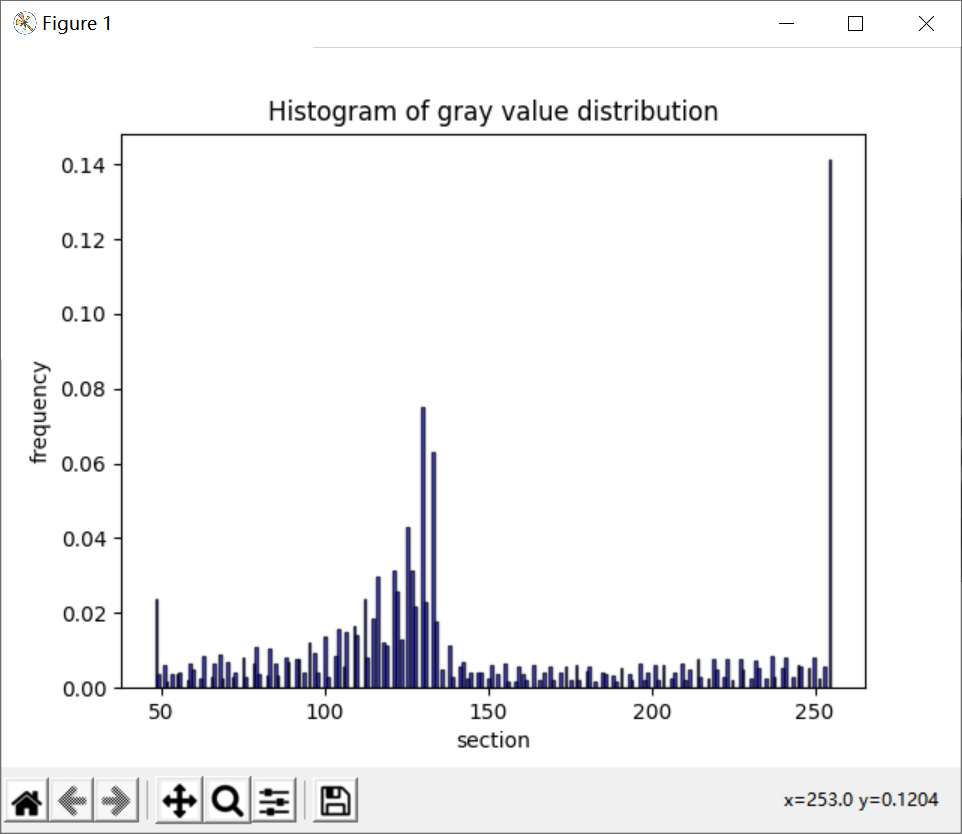
原图直方图:



变换后图象:



变换后直方图:

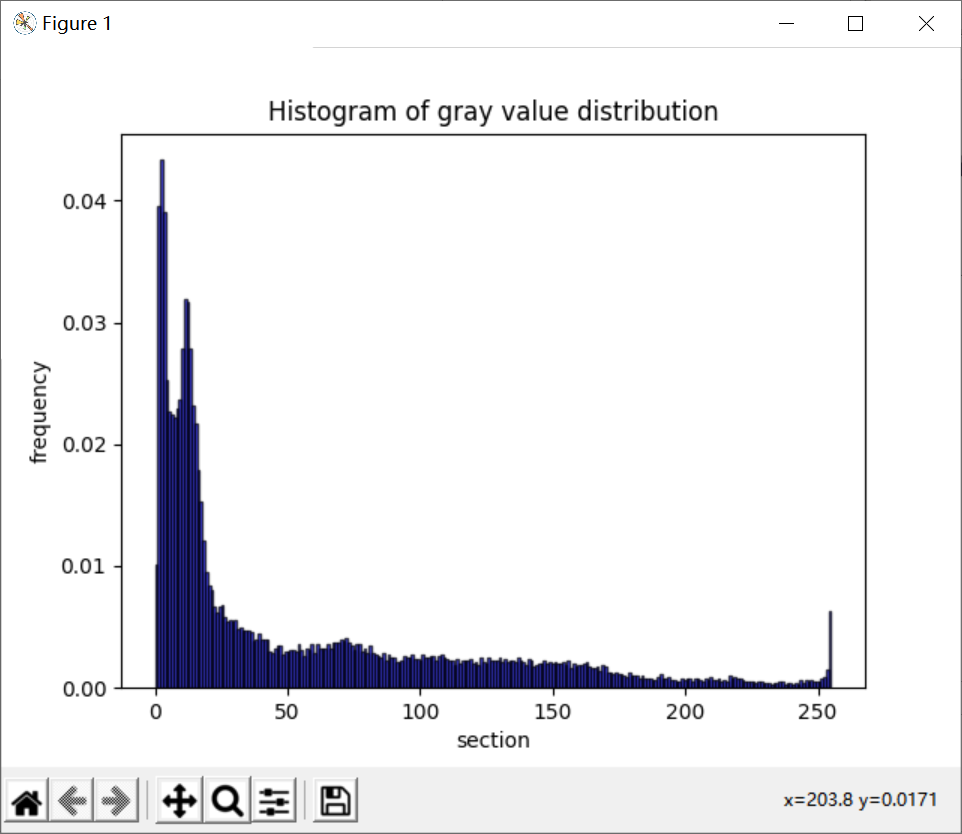


1. 图片3 (c = 0.8, γ = 1.2)

原图:



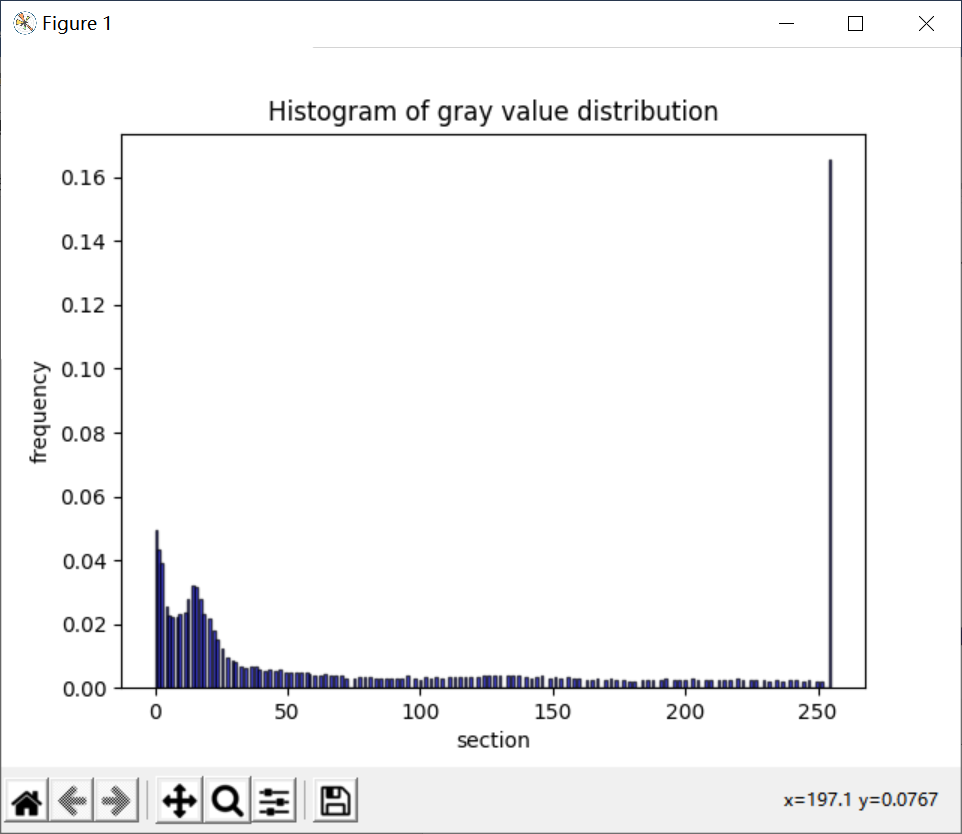
原图直方图:



变换后图象:



变换后直方图:

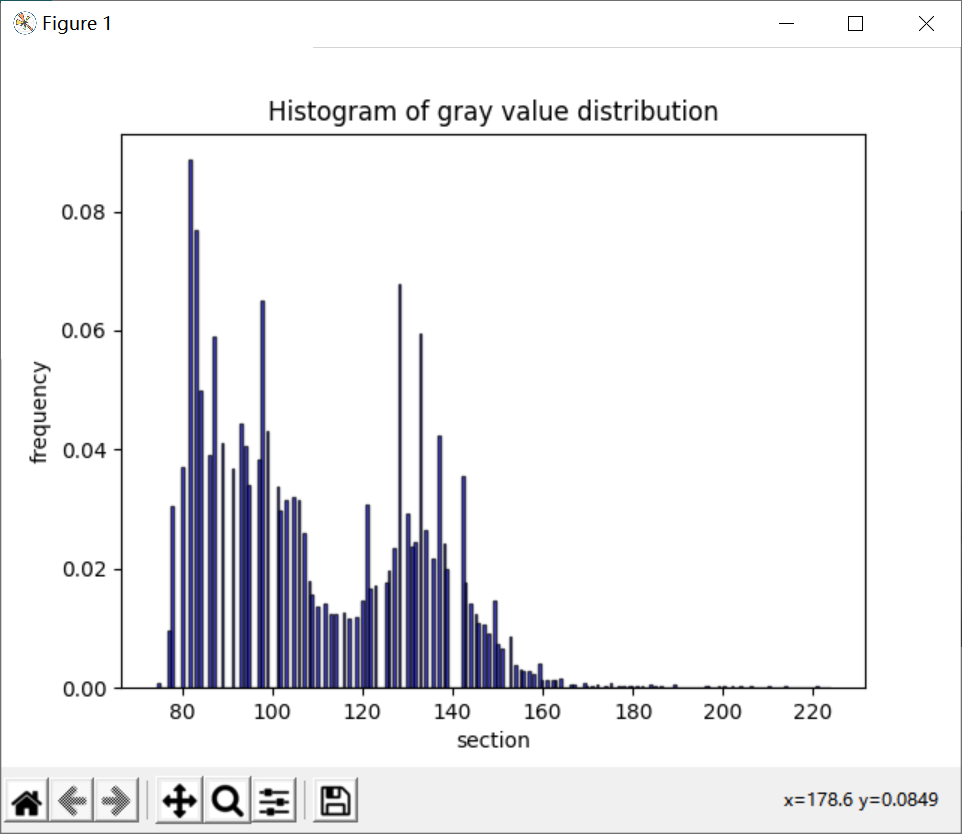


1. **直方图均衡化**
   1. 图片1

原图



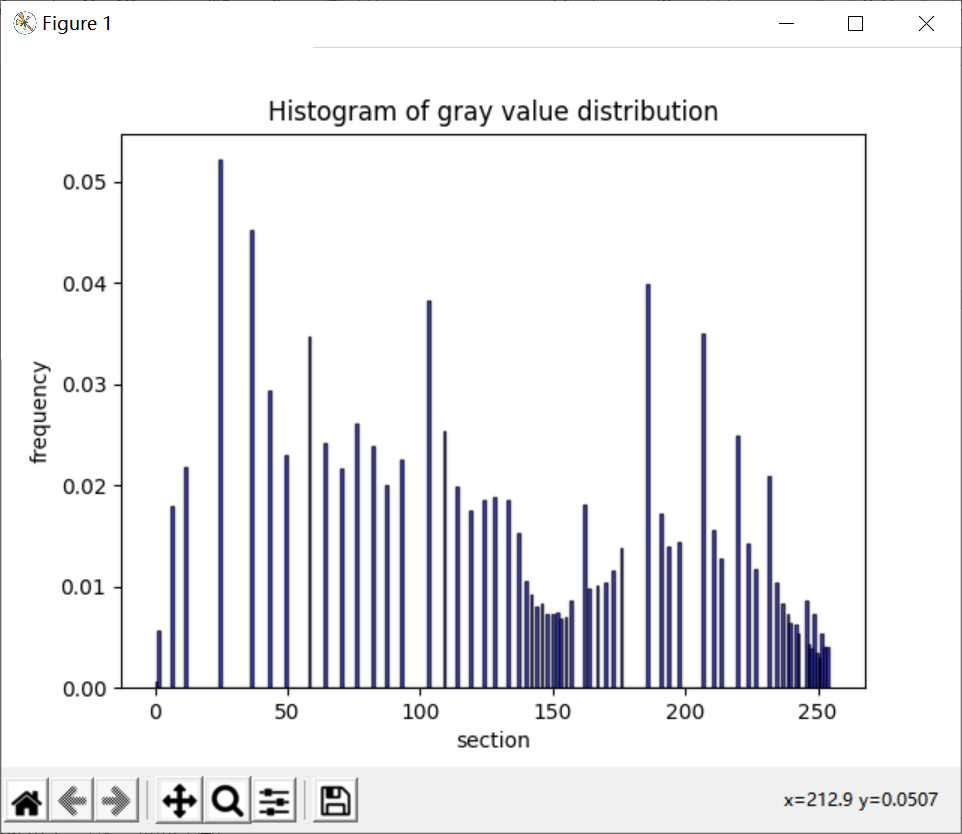
原图直方图:



均衡化后图片

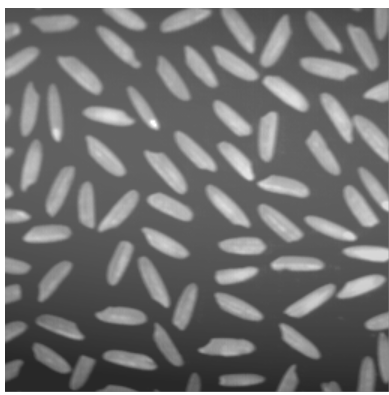


均衡化后直方图

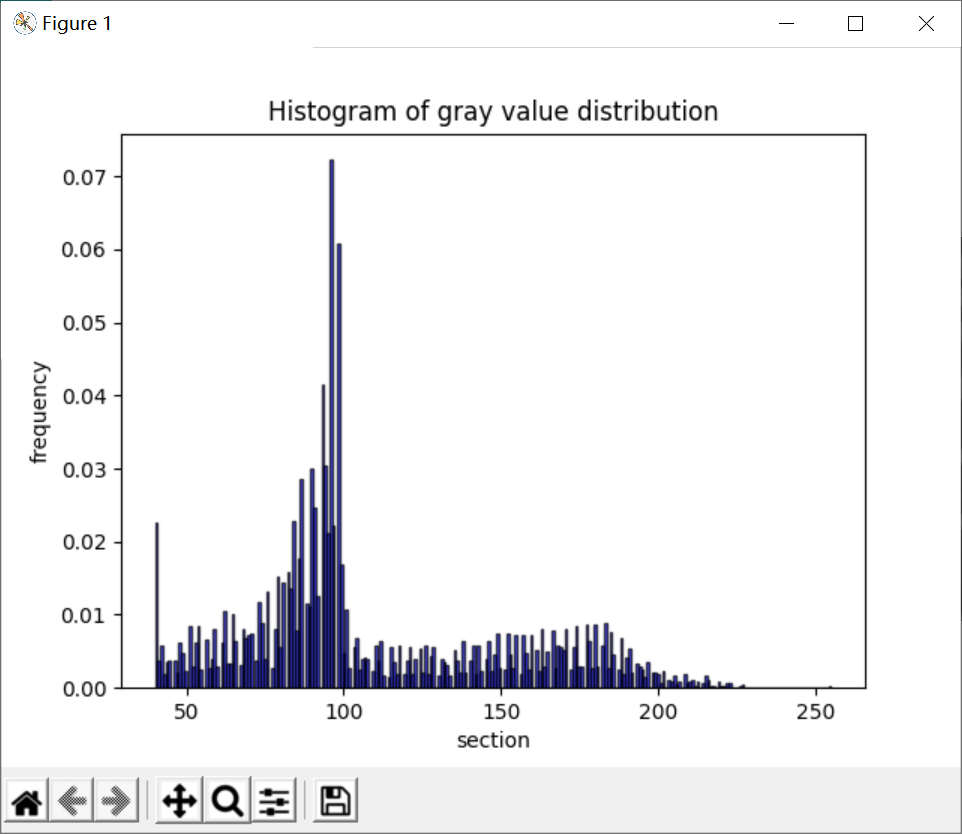


* 1. 图片2

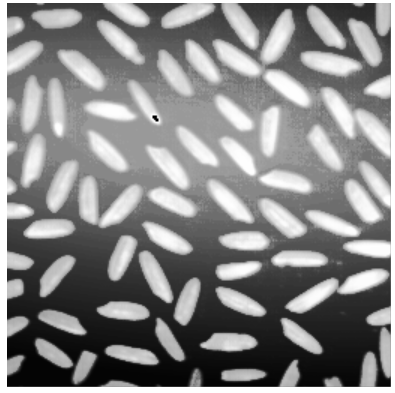
原图



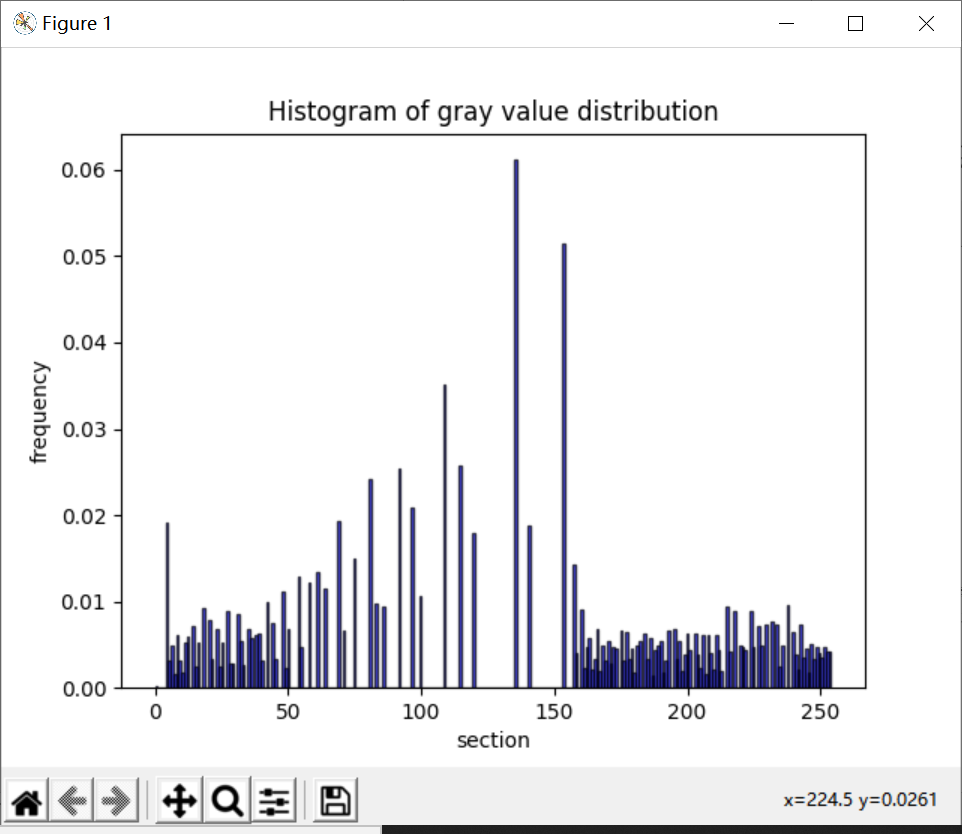
原图直方图



均衡化后图片



均衡化后直方图

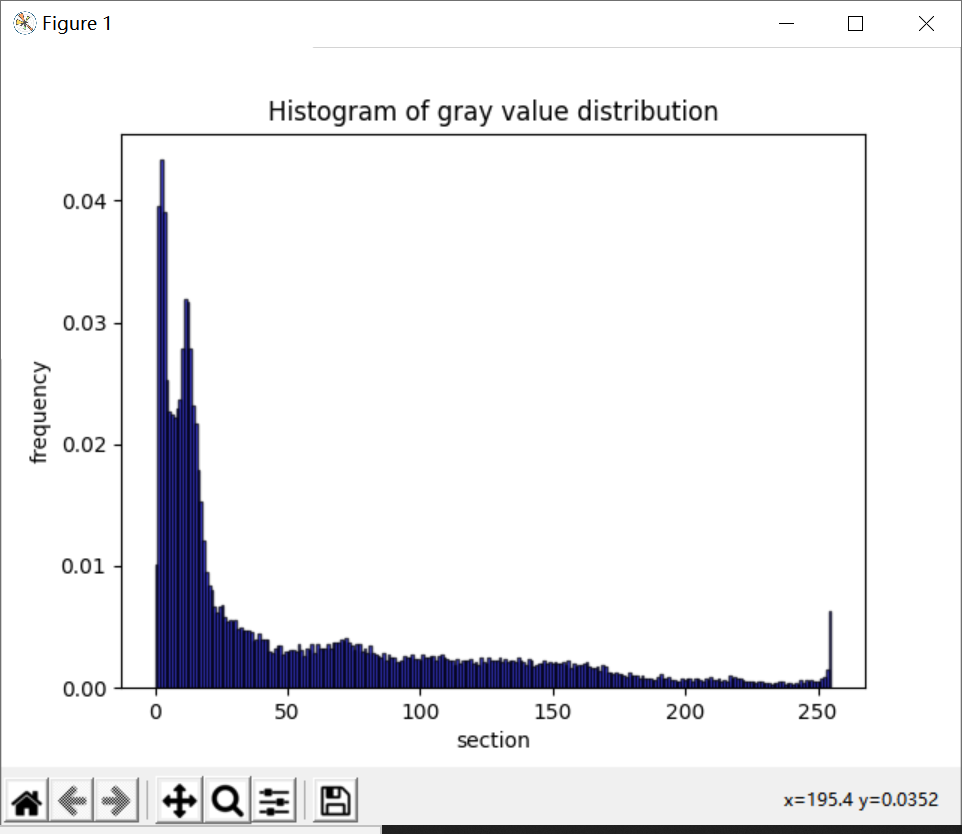


* 1. 图片3

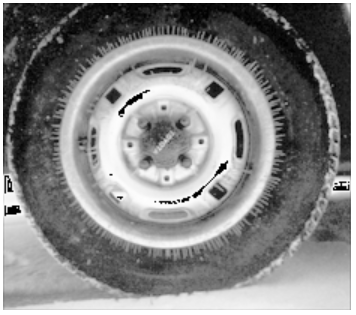
原图



原图直方图



均衡化后图片



均衡化后直方图



1. **附录**

**问题分析**

图像灰度变换是图像处理的基础操作之一，直方图均衡是其中一种方法，用于增强图像的对比度。因此，编制一个程序来实现这两个功能是很有用的。这个程序应该能够读取图像文件，执行灰度变换或直方图均衡，并且能够显示变换后的图像和直方图。

**设计方案**

1. 读取图像文件：使用Python中的OpenCV库或Pillow库中的方法读取图像文件，这些库提供了用于读取和处理图像文件的许多函数和方法。
2. 灰度变换：对于灰度变换，可以使用以下函数对图像进行直接灰度变换：
   * 线性变换：new\_value = (old\_value - min\_value) \* (new\_max - new\_min) / (max\_value - min\_value) + new\_min
   * 对数变换：new\_value = c \* log(1 + old\_value)
   * 幂次变换：new\_value = c \* old\_value ^ gamma 这些变换可以通过选择适当的参数来实现对比度的增强。
   * 本次实验采用对数变换。
3. 直方图均衡：使用以下步骤执行直方图均衡：
   * 计算图像的直方图，即计算每个像素值出现的次数。
   * 计算直方图的累积分布函数（CDF），该函数对应于每个像素值在图像中出现的概率。
   * 计算直方图均衡化函数，该函数将像素值映射到新的像素值。可以使用以下公式进行计算：new\_value = round((L - 1) \* CDF[old\_value])
   * 将图像的每个像素值替换为新的像素值。
4. 显示图像和直方图：程序通过 matplotlib 库显示图像和直方图变化。

**算法**

1. 线性灰度变换和反转

* 1.1. 将图像灰度值减去最大值，然后取反。
* 1.2. 对每个像素进行该操作。

1. 对数变换

* 2.1. 获取图像像素。
* 2.2. 获取图像宽高。
* 2.3. 获取像素最大值。
* 2.4. 对每个像素进行对数变换。
* 2.5. 计算变换后的像素值并将其赋值给新图像。

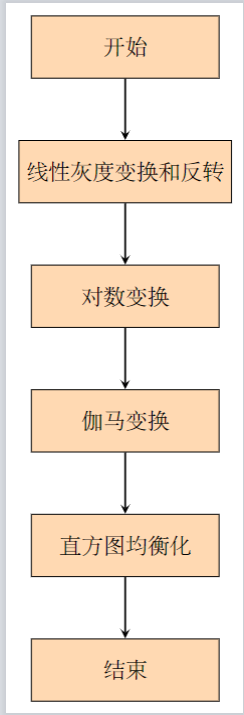
1. 伽马变换

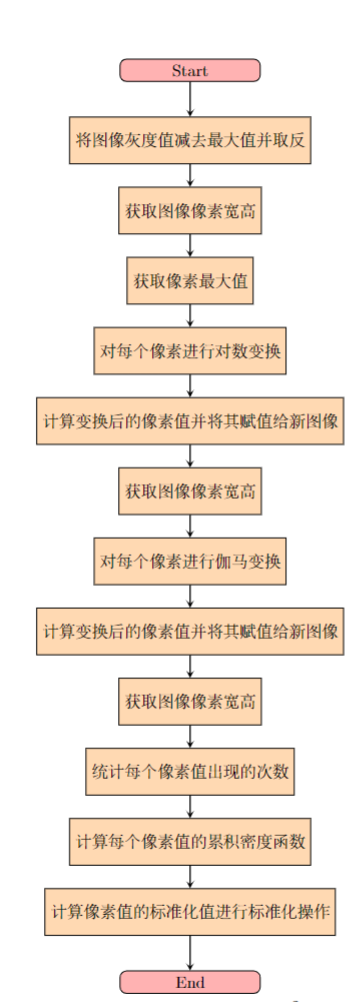
* 3.1. 获取图像像素。
* 3.2. 获取图像宽高。
* 3.3. 对每个像素进行伽马变换。
* 3.4. 计算变换后的像素值并将其赋值给新图像。

1. 直方图均衡化

* 4.1. 获取图像像素。
* 4.2. 获取图像宽高。
* 4.3. 统计每个像素值出现的次数。
* 4.4. 计算每个像素值的累积密度函数。
* 4.5. 计算像素值的标准化值。
* 4.6. 对每个像素进行标准化操作。

**设计图**





源代码

import matplotlib.pyplot as plt  
from PIL import Image  
import numpy as np  
import math  
  
  
# 线性灰度变换，图像反转  
def linearReverse(gray\_image):  
 # 获取输入灰度图像的像素映射和大小  
 gray\_pixel\_map = gray\_image.load()  
 width, height = gray\_image.size  
 mode = gray\_image.mode  
  
 # 创建一个新图像，用于存储变换后的像素值  
 image = Image.new(mode, (width, height))  
 new\_pixel\_map = image.load()  
  
 # 对每个像素进行线性变换和反转  
 for x in range(width):  
 for y in range(height):  
 gray\_pixel = gray\_pixel\_map[x, y]  
 # 线性变换：将像素值减去最大值，然后取反  
 new\_pixel\_map[x, y] = int(255 - (gray\_pixel))  
  
 # 返回变换后的图像  
 return image  
  
# 非线性灰度变换 对数变换  
def logTrans(original\_image):  
 # 加载像素图  
 original\_pixel\_map = original\_image.load()  
 # 获取图像宽高  
 width, height = original\_image.size  
 # 获取图像模式  
 mode = original\_image.mode  
 # 创建一个与原始图像大小和模式相同的新图像  
 new\_image = Image.new(mode, (width, height))  
 # 获取新图像的像素图  
 new\_pixel\_map = new\_image.load()  
 # 找到像素值的最大值  
 max\_value = -1  
 for x in range(width):  
 for y in range(height):  
 if (original\_pixel\_map[x, y] > max\_value):  
 max\_value = original\_pixel\_map[x, y]  
  
 # 对每个像素进行对数变换  
 for x in range(width):  
 for y in range(height):  
 # 获取原始图像上(x, y)坐标处的像素值  
 original\_pixel = original\_pixel\_map[x, y]  
 # 计算变换后的像素值  
 c = 255 / math.log(1 + max\_value)  
 transformed\_pixel = (int)(c \* math.log(original\_pixel + 1))  
 # 将变换后的像素值赋值给新图像  
 new\_pixel\_map[x, y] = transformed\_pixel  
  
 return new\_image  
  
  
# 非线性灰度变换 伽马变换  
def gammaTrans(original\_image):  
 original\_pixel\_map = original\_image.load()  
 width, height = original\_image.size  
 mode = original\_image.mode  
 new\_image = Image.new(mode, (width, height))  
 new\_pixel\_map = new\_image.load()  
  
 for x in range(width):  
 for y in range(height):  
 pixel = original\_pixel\_map[x, y]  
 # 伽马值，值越大图片越暗  
 gamma = 1.2  
 # 调整因子，值越小图片越暗  
 c = 0.8  
 new\_pixel = c \* pow(pixel, gamma)  
 # 防止像素值超出0-255范围  
 if (new\_pixel > 255):  
 new\_pixel = 255  
 new\_pixel\_map[x, y] = (int)(new\_pixel)  
  
 return new\_image  
  
  
  
# 直方图均衡化  
def histogramEqualization(ori\_image):  
 ori\_pixel\_map = ori\_image.load()  
 width, height = ori\_image.size  
 mode = ori\_image.mode  
 size = height \* width  
 image = Image.new(mode, (width, height))  
 new\_pixel\_map = image.load()  
  
 ori\_arr = [0] \* 256  
  
 for x in range(width):  
 for y in range(height):  
 ori\_pixel = ori\_pixel\_map[x, y]  
 ori\_arr[ori\_pixel] += 1  
  
 # for i in range(0,255):  
 # print(str(i) + str(ori\_arr[i]))  
  
 new\_arr = [0] \* 256  
 for i in range(0, 255):  
 new\_arr[i] = ori\_arr[i]  
  
 for i in range(1, 255):  
 new\_arr[i] += new\_arr[i - 1]  
  
 # for i in range(0,255):  
 # print(new\_arr[i])  
  
 for i in range(0, 255):  
 new\_arr[i] = (float)(new\_arr[i] / (size))  
  
 for i in range(0, 255):  
 new\_arr[i] = (int)(new\_arr[i] \* 255)  
  
 # for i in range(0,255):  
 # print(new\_arr[i])  
  
 for x in range(width):  
 for y in range(height):  
 temp = ori\_pixel\_map[x, y]  
 new\_pixel\_map[x, y] = new\_arr[temp]  
  
 return image  
  
  
# 绘制灰度图的直方图  
def drawHistogram(image):  
 width, height = image.size  
 data = []  
 for x in range(width):  
 for y in range(height):  
 gray\_pixel\_map = image.load()  
 gray\_pixel = gray\_pixel\_map[x, y]  
 # print(type(gray\_pixel))  
 pixel = gray\_pixel  
 data.append(pixel)  
  
 plt.hist(data, bins=256, density=0.3, facecolor="blue", edgecolor="black", alpha=0.7, stacked=True)  
 # 显示横轴标签  
 plt.xlabel("section")  
 # 显示纵轴标签  
 plt.ylabel("frequency")  
 # 显示图标题  
 plt.title("Histogram of gray value distribution")  
 plt.show()  
  
  
# 图片的读取  
image = Image.open(r'E:\大三\{选}多媒体技术与应用\2020实验\实验2-图像增强\rice.tif')  
image.show()  
drawHistogram(image)  
  
# 灰度线性变换(反转)  
img = linearReverse(image)  
drawHistogram(img)  
img.show()  
  
# 灰度非线性变换 对数变换  
img = logTrans(image)  
drawHistogram(img)  
img.show()  
  
# 灰度非线性变换 幂律变换(伽马变换)  
img = gammaTrans(image)  
drawHistogram(img)  
img.show()  
  
# 直方图均衡功能  
img = histogramEqualization(image)  
img.show()  
drawHistogram(img)

**程序分析**

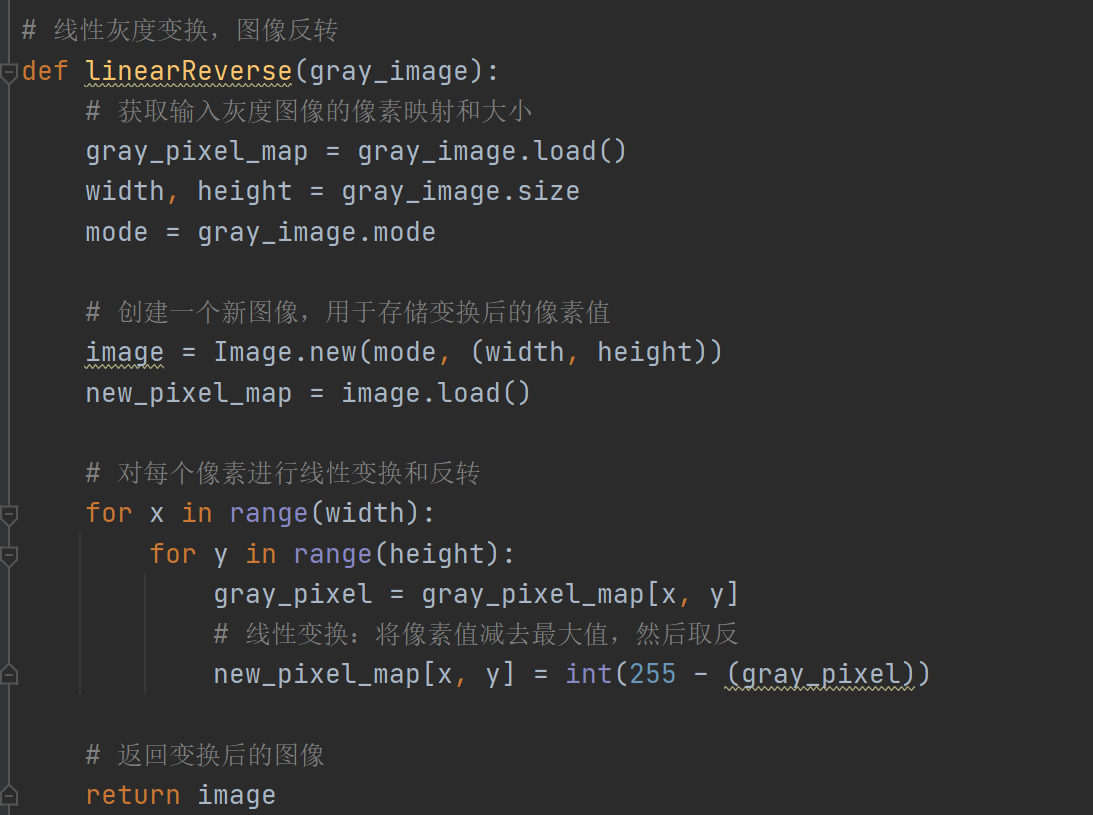
线性灰度变换，即将原来的灰度值按照一定的规则映射到新的灰度值。本代码实现的是图像反转，即将原来的灰度值映射到255-灰度值的新值。

对数变换，是将原来的灰度值进行对数变换后得到新的灰度值。对数变换常用于对暗部细节进行增强，即提高低灰度值的对比度。

伽马变换，是将原来的灰度值进行伽马变换后得到新的灰度值。伽马变换可以用于对图像进行非线性的灰度拉伸和压缩，从而改变图像的对比度和亮度等特性。

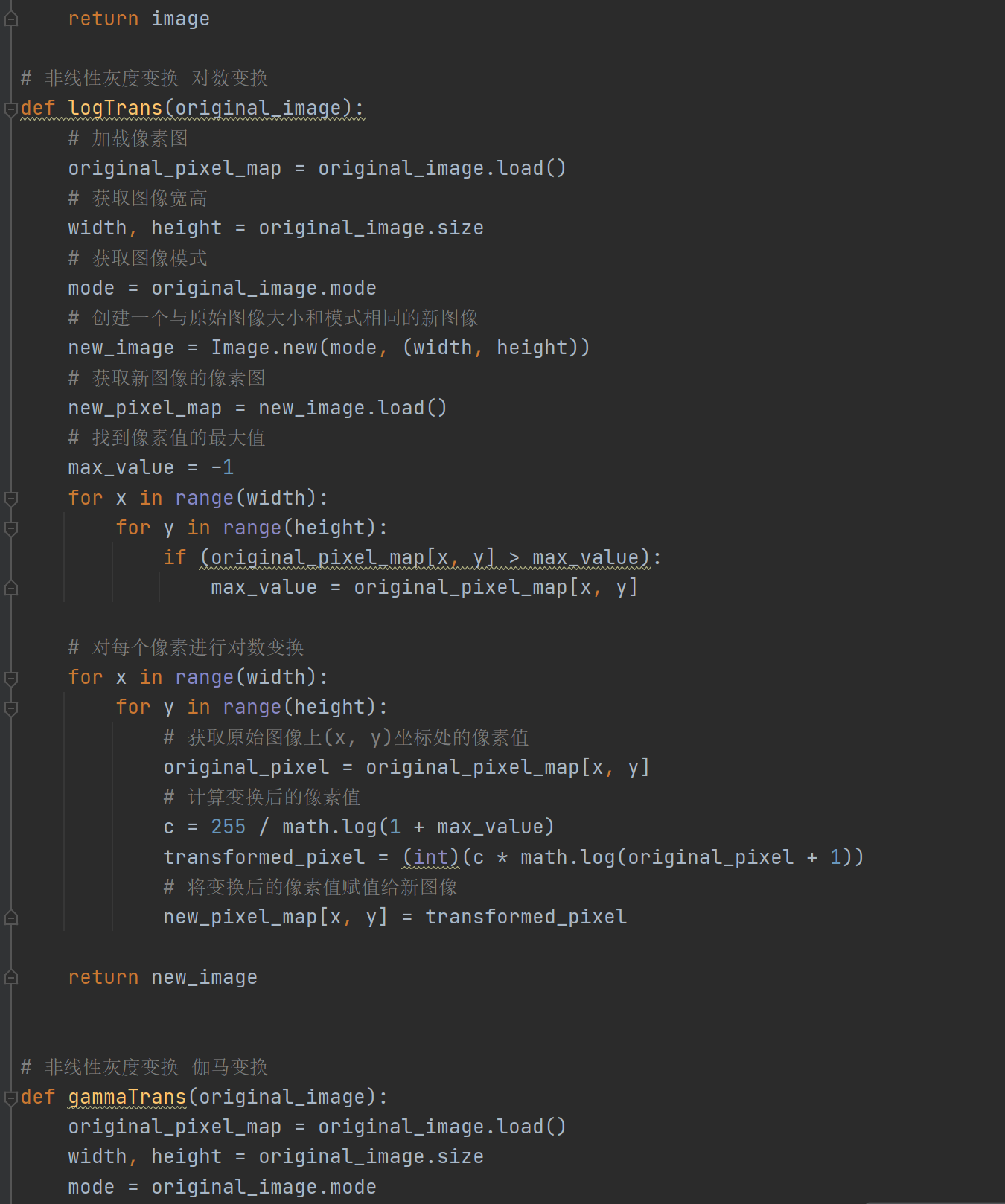
直方图均衡化，是一种用于增强图像对比度的方法。它通过对原始图像的灰度值进行重新分配，使得每个灰度级的像素点数目相等，从而使得图像的对比度更加均衡，细节更加清晰。

1. **图像反转函数**



原理是逐个提取像素的灰度值, 然后利用反转函数反转灰度值. 该函数接受一个灰度图像作为输入，对每个像素进行线性变换和反转，并返回一个新的图像。在函数内部，它首先获取输入图像的像素映射和大小。然后，它创建一个新图像，用于存储变换后的像素值。接下来，它对每个像素进行线性变换和反转。具体地说，它将像素值减去最大值（255），然后取反（用255减去像素值）。最后，它返回变换后的图像。

1. **对数变换函数**



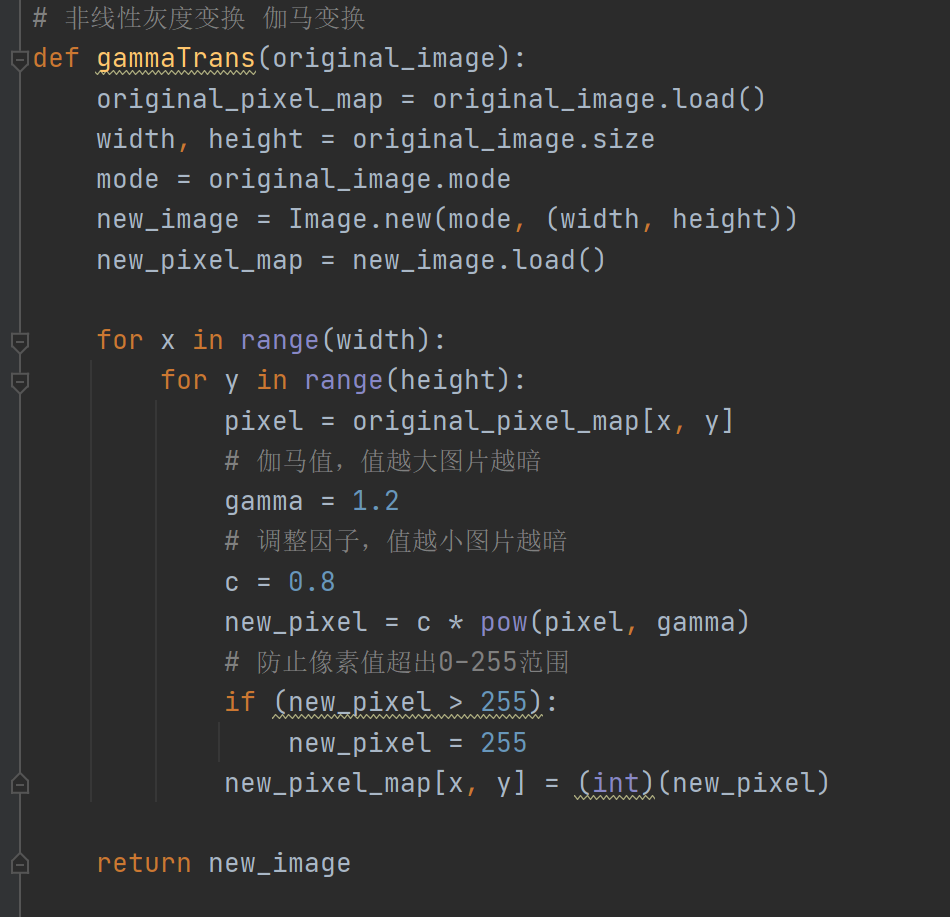
原理是逐个提取灰度值, 之后利用对数变换函数进行变换.

s=c\log (1+r)

其中, 参数c取得是 (255/log(max+1))

其中max是图像中的最大灰度值

1. def logTrans(ori\_image):：定义了一个名为 logTrans 的函数，该函数接受一个原始图像 ori\_image 作为参数。
2. ori\_pixel\_map = ori\_image.load()：获取原始图像的像素点映射表。
3. width, height = ori\_image.size：获取原始图像的宽度和高度。
4. mode = ori\_image.mode：获取原始图像的模式。
5. image = Image.new(mode, (width, height))：创建一个新的图像对象，用来保存变换后的图像。
6. new\_pixel\_map = image.load()：获取新图像的像素点映射表。
7. max = -1：初始化变量 max，用于保存原始图像中最亮的像素值。
8. for x in range(width): for y in range(height): if (ori\_pixel\_map[x, y] > max): max = ori\_pixel\_map[x, y]：循环遍历原始图像的每个像素点，并找到其中的最大像素值，将其保存到变量 max 中。
9. for x in range(width): for y in range(height): pixel = ori\_pixel\_map[x, y] c = 255 / math.log(1 + max) s = (int)(c \* math.log(pixel + 1)) new\_pixel\_map[x, y] = s：循环遍历原始图像的每个像素点，进行对数变换操作。具体地，对于每个像素点的灰度值 pixel，首先计算常数项 c，然后计算对数值 s，最后将 s 作为新图像中对应像素点的灰度值。其中，对数值的计算公式为 s = c \* \log(1 + pixel)，其中 c 是常数项，等于 255 / \log(1 + max)。
10. return image：返回变换后的图像对象。
11. **伽马变换函数**



逐个提取灰度值, 之后利用伽马函数变换灰度值

s=crγ

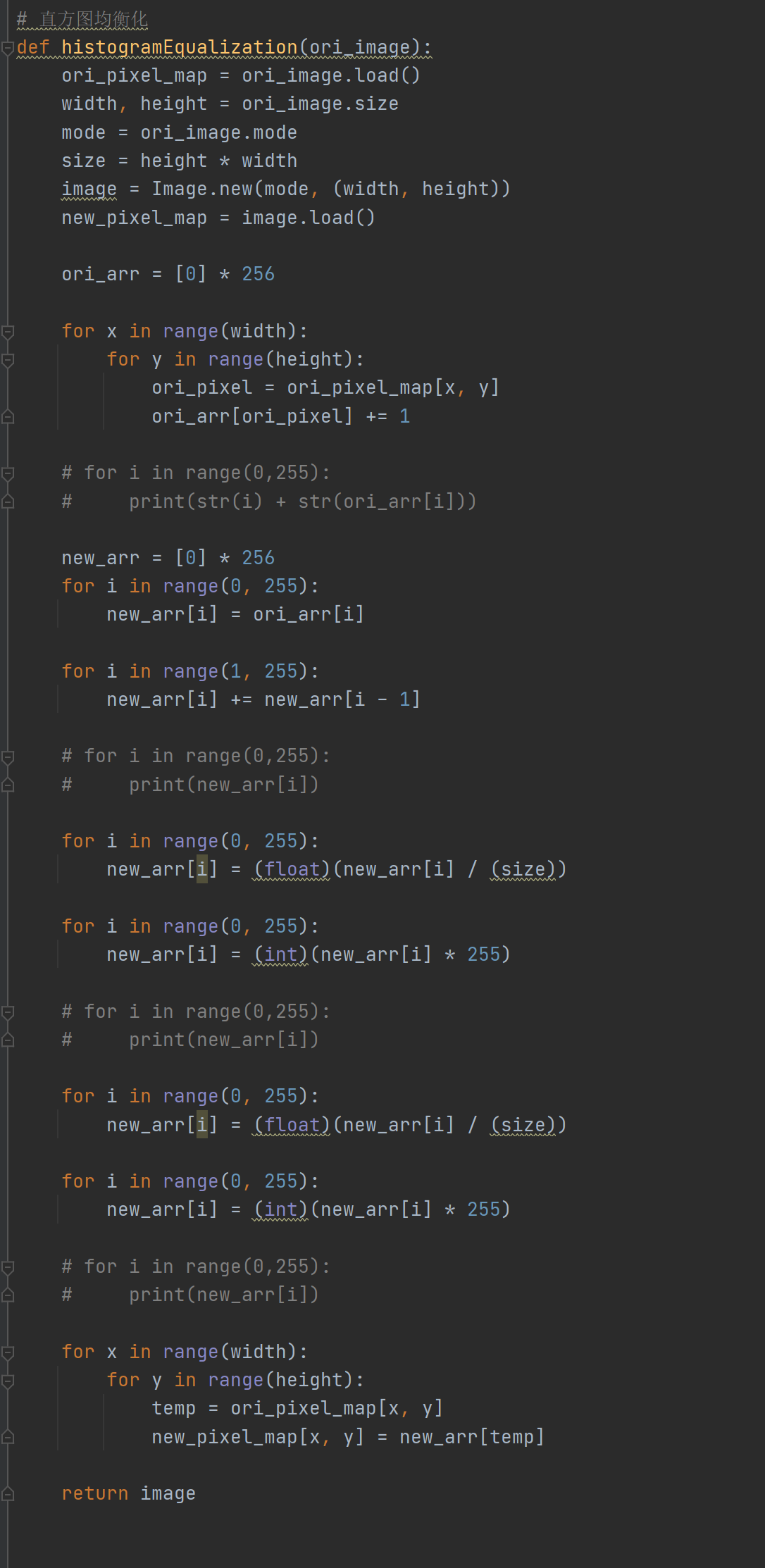
其中, 当γ>1时将较窄范围的低灰度值映射为较宽范围的灰度值，同时将较宽范围的高灰度值映射为较窄范围的灰度值；当γ<1时，情况相反，与反对数变换类似。

参数 c 和 γ 均为实验中手动调参.

代码实现中，首先获取原始图像的像素值信息，并创建一个新的图像对象用于存储变换后的像素值。然后对原图的每个像素进行伽马变换，公式为：new\_pixel = c \* pow(pixel, gamma)，其中c是调整因子，用于控制变换后的像素值范围，gamma是伽马值，用于控制图像亮度和对比度。最后判断变换后的像素值是否超出范围，并将其保存到新图像中。

需要注意的是，伽马值越大，则图像越暗；调整因子越小，则图像越暗。如果伽马值过小或调整因子过大，则会导致图像变得过亮或过暗，影响变换效果。

1. **直方图均衡化函数**



首先计算原始图像的直方图，即统计图像中每个灰度值的像素个数。然后，计算像素在原始直方图中的累计概率分布函数（CDF），并将其映射到一个新的像素灰度值范围（0-255），以获得直方图均衡化后的图像。最后，将原始图像中的每个像素的灰度值替换为其对应的新灰度值即可。

利用直方图均衡化的原理

1. 计算累计直方图；

累加起来就可以

1. 将累计直方图进行区间转换；

使用的是256级灰度

（3）在累计直方图中，概率相近的原始值，会被处理为相同的值。

利用int类型转换

具体实现上，代码首先使用load()方法加载原始图像的像素信息，并创建一个与原始图像大小和模式相同的新图像。然后，通过循环遍历原始图像的所有像素，计算原始图像的直方图，并将其存储在一个长度为256的数组中。接下来，算法通过计算原始直方图的累计分布函数（CDF），并将其映射到新的像素灰度值范围，计算新的灰度映射数组。最后，代码通过遍历原始图像的每个像素，并将其灰度值映射到新的灰度值，生成直方图均衡化后的图像。

1. **绘制直方图函数**

**同上次实验**

**实验心得**

在实验中，我深刻理解了直接灰度变换和直方图均衡的原理，了解了如何通过调整图像的灰度值来增强图像的对比度。同时，我也掌握了直接灰度变换和直方图均衡技术的具体实现方法，包括使用Python代码进行实现的方法。

在实验中，我通过使用Python语言来实现直接灰度变换和直方图均衡技术。我发现，Python的简洁性和灵活性使得实现这些技术变得非常容易。同时，通过这个实验，我也加深了我对Python的理解，并提高了我的Python编程能力。