实验目的

掌握基本图像增强算法的实现

实验内容

- 直方图均衡化图像增强
- 对数图像增强
- 幂次图像增强

算法设计

直方图均衡化图像增强

①对图像遍历，统计出每个像素的出现次数（即频率），形成直方图

②对所有频率求和

③对直方图的每个频率除以所有频率的和，求出归一化到 0-255 的范围

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

对数图像增强

s = c * log(1+r)

幂次图像增强

s = c * pow(r,γ)

代码

```
# 直方图均衡化

from PIL import Image

def read_image(file_path):
    # 使用PIL 库读取图像
    image = Image.open(file_path).convert("L")  # 转为灰度图像，L 代表灰度模式
    return image
```

```python
def histogram_equalization(image):
    # 获取图像的直方图
    histogram = [0] * 256
    width, height = image.size

    for y in range(height):
        for x in range(width):
            pixel_value = image.getpixel((x, y))
            histogram[pixel_value] += 1

    # 计算累积分布函数 列表
    cumulative_distribution = [sum(histogram[:i+1]) for i in range(256)]

    # 归一化到0-255 的范围
    normalized_cdf = [int(cumulative_distribution[i]/cumulative_distrib
ution[-1] * 255) for i in range(256)]

    # 应用均衡化
    equalized_image = Image.new("L", (width, height))

    for y in range(height):
        for x in range(width):
            pixel_value = image.getpixel((x, y))
            equalized_value = normalized_cdf[pixel_value]
            equalized_image.putpixel((x, y), equalized_value)

    return equalized_image

def main():
    # 读取图像
    input_path = "../clear.bmp"
    original_image = read_image(input_path)

    # 进行直方图均衡化
    equalized_image = histogram_equalization(original_image)

    # 显示原始图像和均衡化后的图像
    original_image.show(title="Original Image")
    equalized_image.show(title="Equalized Image")

    # 保存均衡化后的图像
    output_path = "../historgram_result/clear_my.bmp"
    equalized_image.save(output_path)

if __name__ == "__main__":
    main()
```

```python
# 对数强化


from PIL import Image
import math

def read_image(file_path):
    # 使用PIL库读取图像
    image = Image.open(file_path).convert("L")  # 转为灰度图像，L 代表灰度模式
    return image

def log_transform(image, c, base):
    width, height = image.size

    equalized_image = Image.new("L", (width, height))

    for y in range(height):
        for x in range(width):
            pixel_value = image.getpixel((x, y))

            # 对数变换t
            log_transformed_value = c * math.log10(pixel_value + 1) / math.log10(base)
            # print(log_transformed_value)

            # 将变换后的值限制在[0, 255]的范围内
            log_transformed_value = (log_transformed_value/ math.log10(base))*255

            equalized_image.putpixel((x, y), int(log_transformed_value))


    return equalized_image

def main():
    # 读取图像
    input_path = "../lab1-7.tif"
    original_image = read_image(input_path)

    base = 150

    # 进行直方图均衡化
    equalized_image = log_transform(original_image,1,base)

    # 显示原始图像和均衡化后的图像
    original_image.show(title="Original Image")
```

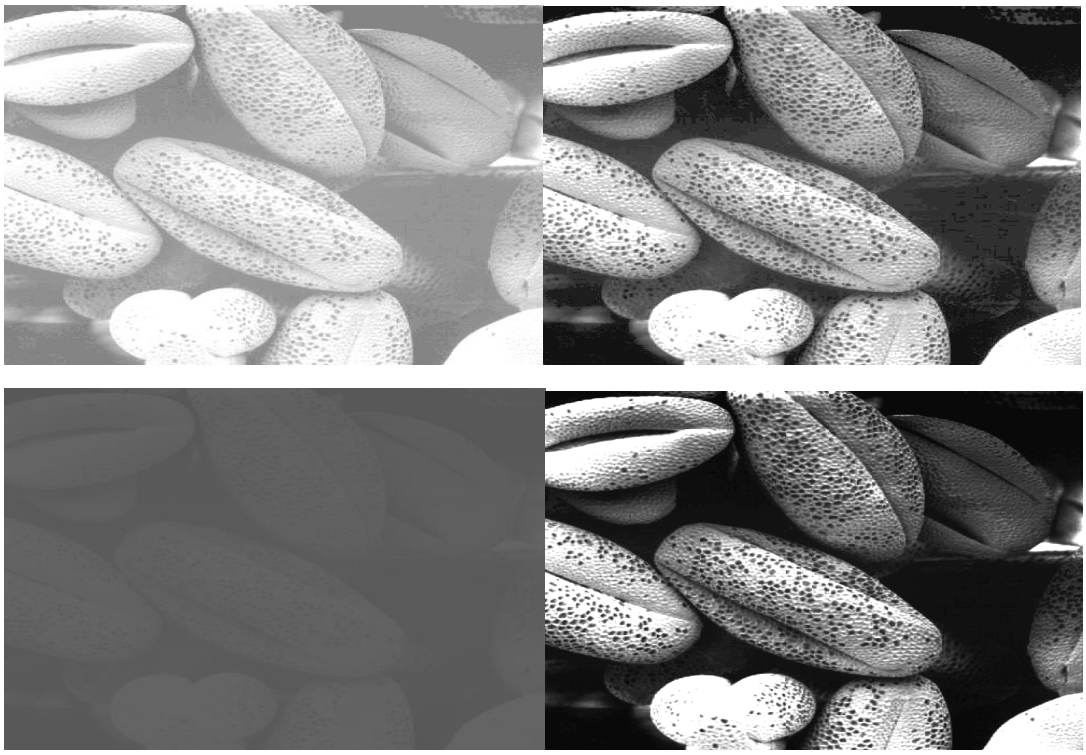
```
    equalized_image.show(title="Equalized Image")

    # 保存均衡化后的图像
    output_path = "../log_transform/lab1-7_my_"+str(base)+"_.tif"
    equalized_image.save(output_path)

if __name__ == "__main__":
    main()

# 幂次强化



from PIL import Image
import math

def read_image(file_path):
    # 使用PIL库读取图像
    image = Image.open(file_path).convert("L")  # 转为灰度图像，L 代表灰
度模式
    return image

def log_transform(image, c, base):
    width, height = image.size

    equalized_image = Image.new("L", (width, height))

    for y in range(height):
        for x in range(width):
            pixel_value = image.getpixel((x, y))

            # 幂次变换
            log_transformed_value = c * math.pow(pixel_value,base)
            # print(log_transformed_value)

            # 将变换后的值限制在[0, 255]的范围内
            log_transformed_value = (log_transformed_value/ (math.pow(2
55,base)))*255

            equalized_image.putpixel((x, y), int(log_transformed_value))

    return equalized_image

def main():
    # 读取图像
    input_path = "../unclear.bmp"
    original_image = read_image(input_path)
```

```
    base = 0.05

    # 进行直方图均衡化
    equalized_image = log_transform(original_image,1,base)

    # 显示原始图像和均衡化后的图像
    original_image.show(title="Original Image")
    equalized_image.show(title="Equalized Image")

    # 保存均衡化后的图像
    output_path = "../power_transform/unclear_my_"+str(base)+"_.bmp"
    equalized_image.save(output_path)

if __name__ == "__main__":
    main()
```

实验结果

说明：第一行左边为原图，右边为直方图均衡化后的图；第二行左边为 log 变换后的图，右边为幂次变换后的图，参数会在对应位置说明

下图 log 变换底数为 300，幂次变换指数为 5

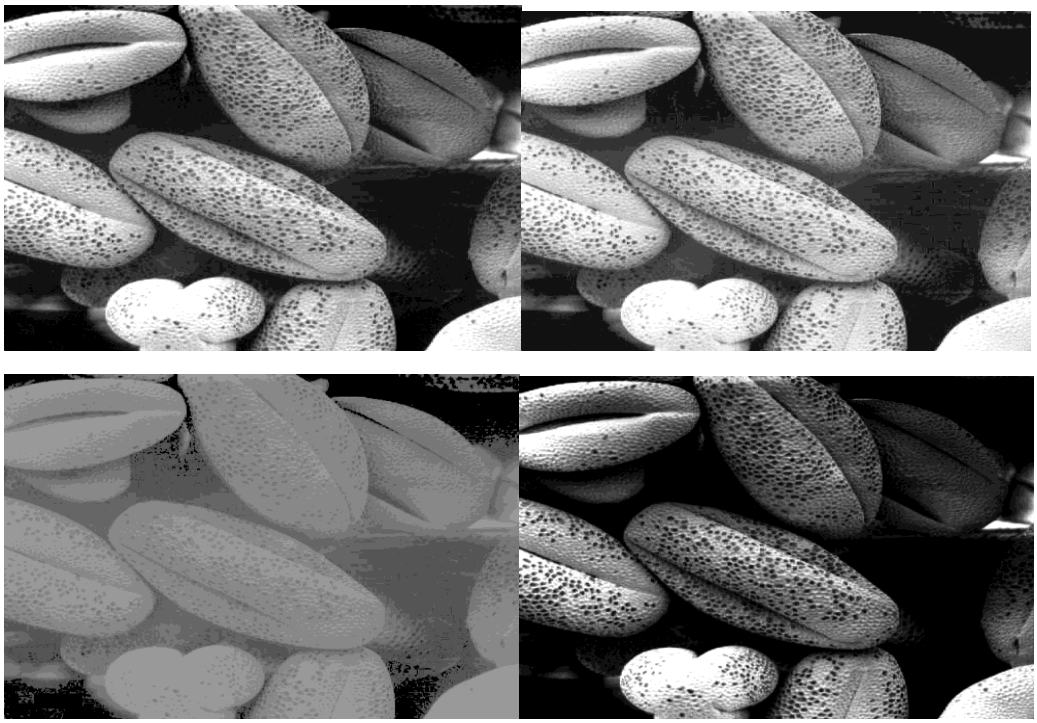


原图为过亮的图片，所以在对图片取较小数值为 log 底数的时候，得到更亮的图片，于是我尝试将底数剧增，当底数达到一定程度（比如 300）的时候，起到了调暗的作用，但是也更模糊。

对于均衡化强化，效果比 log 好，因为增强了对比度，黑白色之间的差值变大，显得清晰了很多

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 5，得到了一个较原图暗的结果，也很清晰。

下图 log 变换底数为 100，幂次变换指数为 2

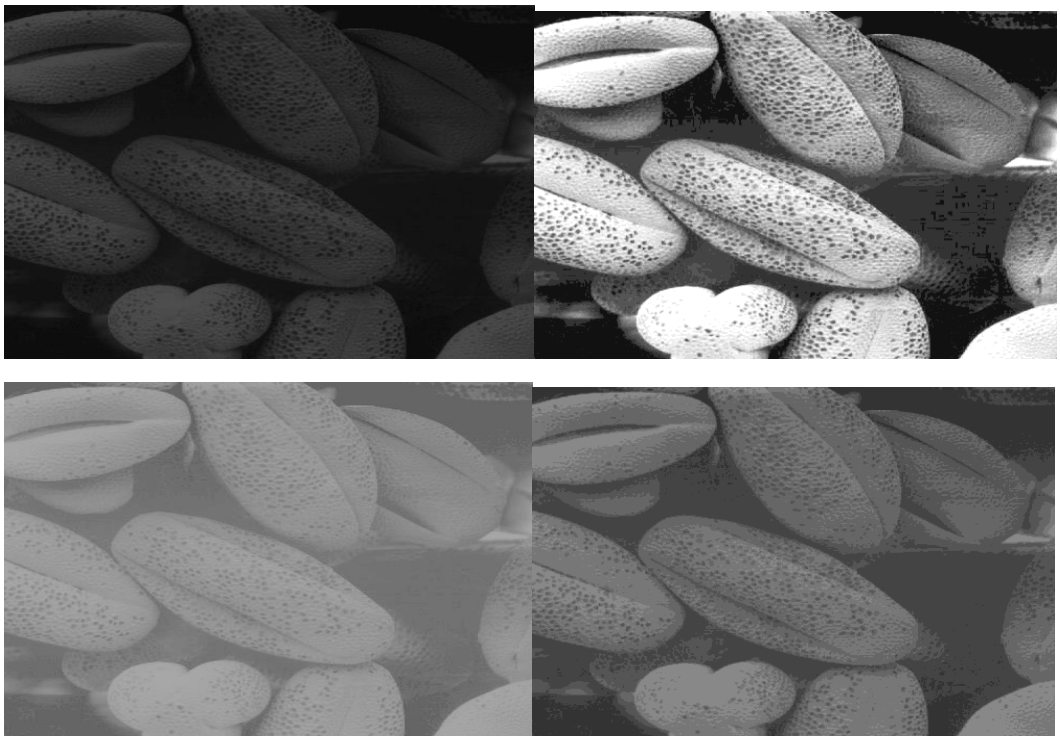


原图为正常的图片，所以在对图片取较小数值为 log 底数（比如 30）的时候， 会得到非常亮的图片，于是与上面的图片一样，只是这次取 100 就已经比较模糊比较暗了

对于均衡化强化，效果比 log 好，因为增强了对比度，黑白色之间的差值变大，会比原图稍微亮一点点

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 2，得到了一个较原图暗的结果，也很清晰。

下图 log 变换底数为 50，幂次变换指数为 0.5

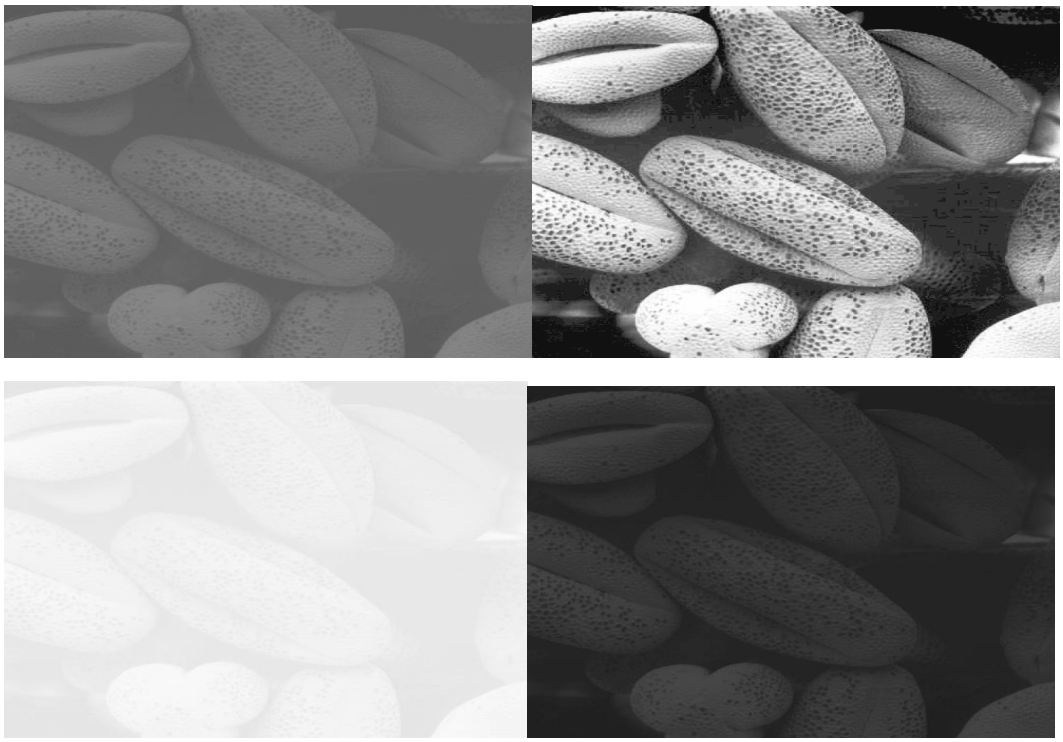


原图为过暗的图片，在对图片取底数为 50 的变换，图片稍微亮了一点，当取 15 的时候（未贴图），图片过于明亮，显得刺眼

对于均衡化强化，效果比 log 好，因为增强了对比度，黑白色之间的差值变大，显得清晰了很多

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 0.5，得到了一个较原图亮的结果，也比较模糊，但是比 log 变换效果好。

下图 log 变换底数为 50，幂次变换指数为 2

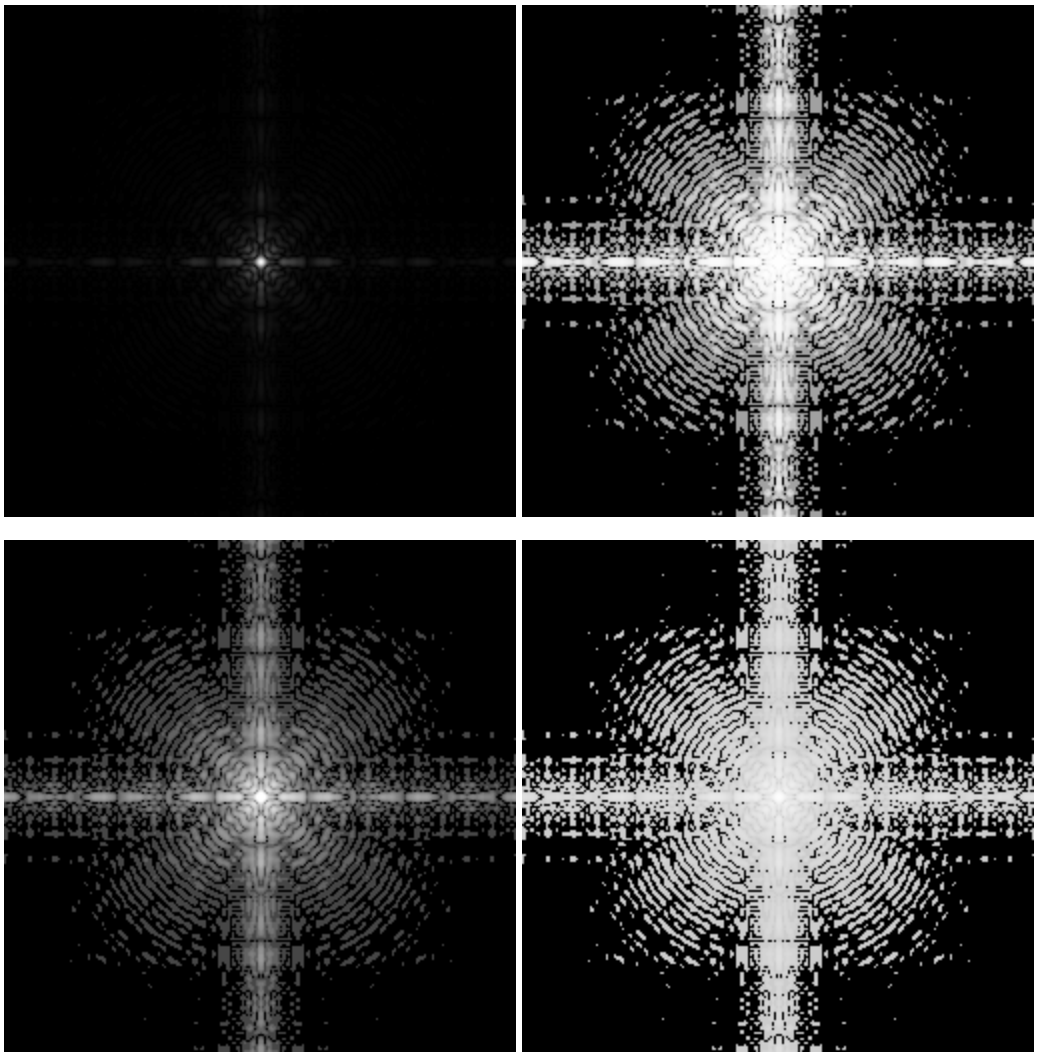


原图为灰蒙蒙的图片，在对图片作 log 变换时，总是不清楚的，因为 log 变换是调整整体亮度，并不能像直方图均衡化那样拉开对比度，此处取对数为 50。

对于均衡化强化，效果比 log 好，因为增强了对比度，黑白色之间的差值变大，显得清晰了很多，去除了灰蒙蒙的效果

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 2，得到了一个较原图暗的结果，也很模糊，并且与 log 变换一样，不能提高对比度，是整体亮度调整。

下图 log 变换底数为 30，幂次变换指数为 0.05

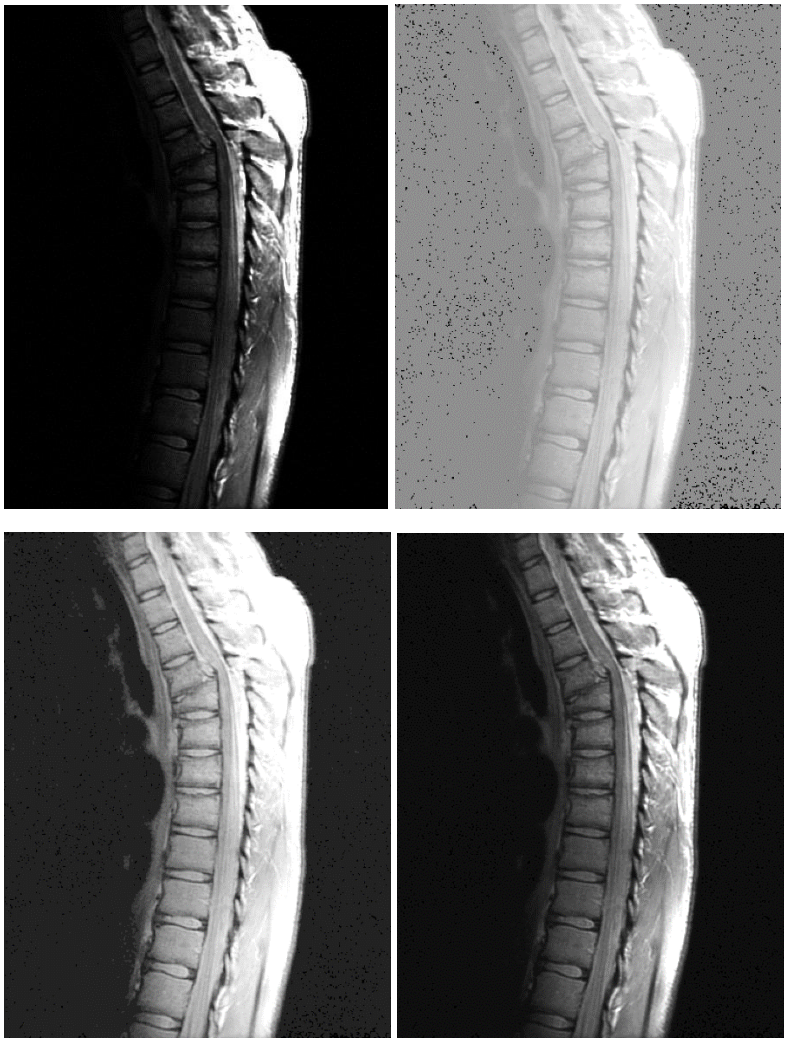


原图为过暗的图片，在对图片作 log 变换，取底数为 30 时得到了一个较为清晰的图片。

对于均衡化强化，效果比 log 清晰一些，因为增强了对比度，黑白色之间的差值变大，显得清晰了很多

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 0.05，得到了一个较原图亮的结果，也很清晰。

下图 log 变换底数为 30，幂次变换指数为 0.5



原图为过暗的图片，在对图片作 log 变换，取底数为 30 时得到了一个较为清晰的图片。

对于均衡化强化，效果比 log 差一些，因为增强了对比度，拉伸了黑色像素占据的空间，使得整幅图像都有些模糊。

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 0.5，得到了一个较原图亮的结果，也很清晰。

三种变换都比原图在细节上体现得更加清楚

下图 log 变换底数为 150，幂次变换指数为 5

原图为过亮的图片，在对图片作 log 变换，取底数为 150 时得到了一个较暗的图片，而且非常模糊

对于均衡化强化，效果比 log 好，因为增强了对比度，黑白色之间的差值变大，显得清晰了很多

对于幂次变换，当 γ 大于 1 的时候，此处 γ 取 05，得到了一个较原图暗的结果，也很清晰。

结论

已经在每幅图下边注明