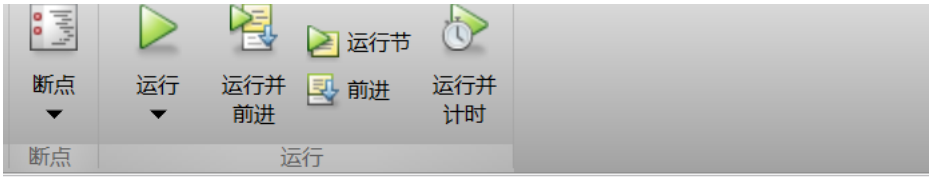
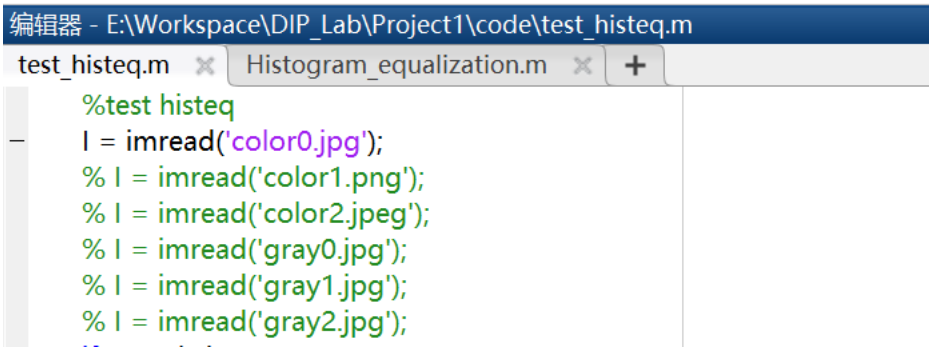
数字图像处理 实验一

一、实验完成情况概述

- 完成了函数 **hist_equal** 的实现，用以进行灰度图像直方图均衡化；
- 除了框架代码给出的方法，还另外实现了两种方法来进行 RGB 图像的均衡化；
- 共使用了六张图片来验证方法的有效性。

代码运行方式：





如图所示，直接在 Matlab 图形界面点击运行即可，可通过修改下方注释内容读取不同的图片。

二、灰度图像的直方图均衡化实现

根据课堂上所讲的理论内容，**hist_equal** 函数的实现是比较容易的。

所用到的核心公式就是

$$f(D_A) = \frac{L}{A_0} \sum_{u=0}^{D_A} H_A(u)$$

对于默认的情况，只需要传入所需要进行直方图均衡化的图像矩阵。程序会调用 **imhist** 函数得到表示传入图像的灰度直方图的向量 ***H***，接着通过循环计算向量 ***H*** 的前缀和，记录在 ***sum*** 向量中。***H*** 和 ***sum*** 均是尺寸为 256*1 的列向量。那么，在上面的公式中，对于取值为 0-255 的灰度值D_A，$\sum_{u=0}^{D_A} H_A(u)$的值就可以通过 ***sum(D_A+1)***来获得了（由于 Matlab 是下标索引是从 1 开始的，故需要加一）。再乘上系数$\frac{L}{A_0}$，即可得到最终的变换向量 ***F***，它能将原图像中的灰度值映射到新的灰度值，从而实现图像的均衡化。

如果在调用 **hist_equal** 函数时还同时传入一个给定的直方图，那么程序将通过直接传入的直方图来计算变换向量，而不是通过原图像的直方图来计算。这在后面对 RGB 图像进行均衡化时会用到。

代码展示如下：

```
function [output2] = hist_equal(input_channel, varargin)
    [row, col] = size(input_channel);
    if nargin == 1
        H = imhist(input_channel);
    elseif nargin == 2
        H = varargin{1};
    end
    sum = zeros(256, 1);
    for i = 1 : 256
        if(i == 1)
            sum(i) = H(i);
        else
            sum(i) = sum(i - 1) + H(i);
        end
    end
    F = 255 * sum / (row * col);
    output2 = zeros(row, col);
    for i = 1 : row
        for j = 1 : col
            output2(i, j) = round(F(input_channel(i, j) + 1));
        end
    end
    output2 = uint8(output2);
end
```

三、RGB 图像均衡化的三种方法

（1） 方法一

第一种方法是最简单直白的方法：RGB 图像是由 R,G,B 三个颜色通道叠加而成的，每个通道都可以看作是一个灰度图像，因此可以采用对三个颜色通道分别均衡化的方法来对 RGB 图像进行均衡化。

方法一的代码在原框架代码中已经给出，在实现了灰度图像均衡化函数 **hist_equal** 之后，这一方法可以直接完成。

（2） 方法二

第二种方法仍然是对三个颜色通道的灰度值进行操作。不同的是，这里对三个通道进行

均衡化时使用的不是它们各自的直方图，而是三个通道的直方图的均值。**hist_equal** 的实现为此提供了方便，只需要将均值直方图作为第二个参数分别传入即可。

（3） 方法三

第三种方法与前两种不同。通过查询相关的资料，我了解到可以将 RGB 图像转化到 HSV 颜色空间中，并对明度(V)通道进行均衡化。

得益于 Matlab 库函数 **rgb2hsv** 和 **hsv2rgb** 的支持，这一方法的实现也不复杂。只是需要注意的是，V 通道的取值是 0-1 之间的 double 类型的浮点数，要调用 **hist_equal** 函数对其进行均衡化，需要将其线性映射到 0-255 之间的 uint8 类型的整数。

三种方法的代码展示如下：

```
r=input_image(:,:,1);
g=input_image(:,:,2);
b=input_image(:,:,3);
switch varargin{1}
    case 1
        r1 = hist_equal(r);
        g1 = hist_equal(g);
        b1 = hist_equal(b);
        output = cat(3,r1,g1,b1);
    case 2
        Hr = imhist(r);
        Hg = imhist(g);
        Hb = imhist(b);
        H2 = round((Hr + Hg + Hb) / 3);
        r2 = hist_equal(r, H2);
        g2 = hist_equal(g, H2);
        b2 = hist_equal(b, H2);
        output = cat(3,r2,g2,b2);
    case 3
        hsvimg = rgb2hsv(input_image);
        v = hsvimg(:,:,3);
        v = uint8(v * 255);
        v = hist_equal(v);
        v = im2double(v);
        output = hsvimg;
        output(:,:,3) = v;
        output = hsv2rgb(output);
end
```

四、结果展示与方法比较

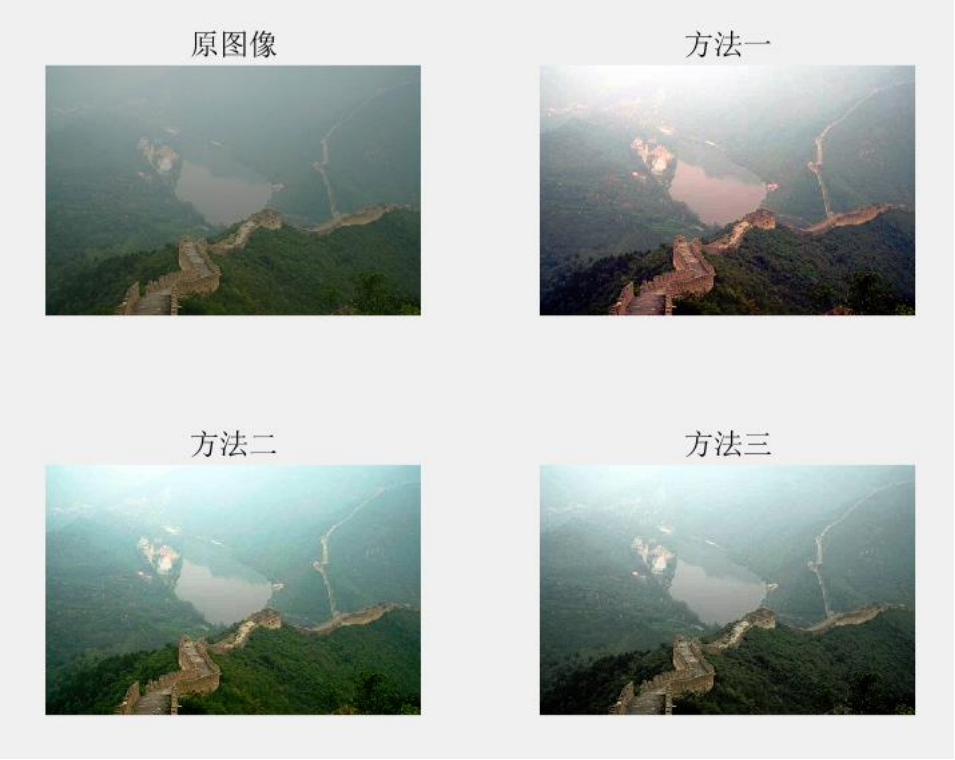
（1） 灰度图像均衡化效果

gray0.jpg

gray1.jpg

gray2.jpg

（2） 彩色图像均衡化效果



color0.jpg

color1.jpg

color2.jpg

（3） 方法比较

在三种对彩色图像进行均衡化的方法中，我们可以清楚地看到，尽管方法一实现简单，但是却带来比较严重的色彩失真。而方法二和方法三则在不同的图片上表现各有优劣。在第二幅彩色图片上，使用方法二进行处理能够更好地还原真实图片的色彩；而在第三幅彩色图片上，使用方法三得到的图片具有更高的对比度，能更好地展示出云层和山脉的细节。