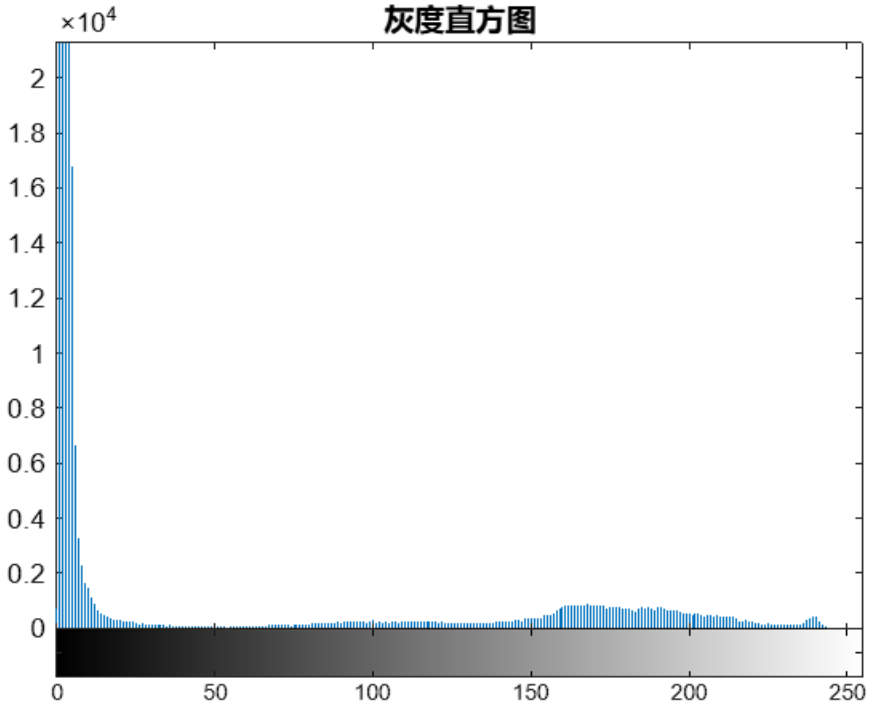
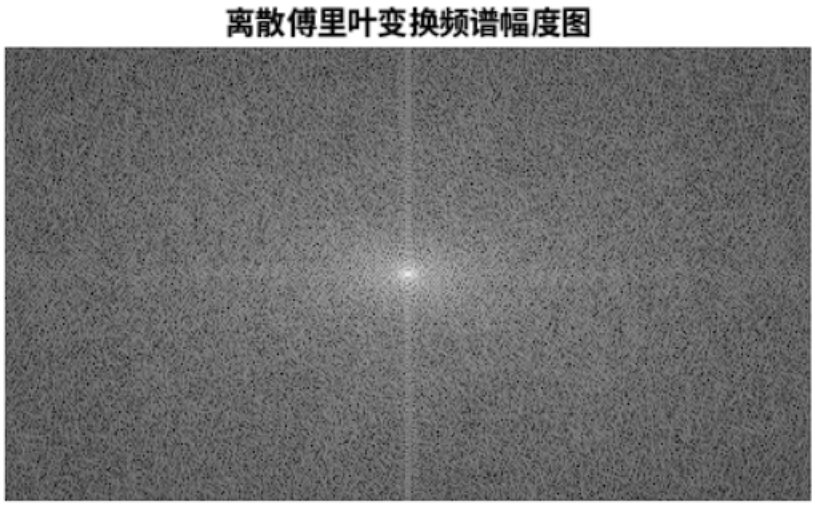
1. 实验一：灰度直方图和傅里叶变换频谱幅度图

1.1. Matlab 代码

```
input_image = imread('低照度图像.jpg')
%将彩色图像转换为灰度图
gray_image = rgb2gray(input_image)
%输出灰度图
%figure
%imshow(gray_image)
%title("灰度图")
%转换为灰度直方图
figure
imhist(gray_image)
title("灰度直方图")
%求离散傅里叶变换频谱幅度图
fft2_image = fft2(gray_image)
fft2_image_shifted = fftshift(fft2_image)
%计算幅度
magnitude = abs(fft2_image_shifted)
figure
imshow(log(1+magnitude),[])
title('离散傅里叶变换频谱幅度图')
```

1.2. 运行结果


灰度直方图
×10^4
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
50
100
150
200
250


离散傅里叶变换频谱幅度图

2. 实验二：直方图均衡化和同态滤波

2.1. Matlab 代码

```
input_image = imread('低照度图像.jpg')
%转换为灰度图
gray_image = rgb2gray(input_image)
%进行直方图均衡化操作
average_image = histeq(gray_image)
%进行同态滤波操作
%将图像转换为浮点数并进行对数变换
log_image = log(double(gray_image)+1)
%设计高通滤波器
[M,N] = size(log_image) %获取图像的大小
[x,y] = meshgrid(1:N,1:M) %创建网络坐标，便于计算频率
D0 = 30 %设置截止频率
D = sqrt((x-N/2).^2+(y-M/2).^2) %计算每个点到频谱中心的距离
H = exp(-D.^2/(2*(D0^2))) %生成高通滤波器的频率响应
%进行傅里叶变换，应用滤波器
F = fft2(log_image) %对对数变换后的图像进行傅里叶变换
F_flitered = F.*(1-H) %应用高通滤波器，保留高频成分
flitered_image = real(ifft2(F_flitered)) %对滤波结果进行反傅里叶变换，并取实部
flitered_image = exp(flitered_image)-1 %对结果进行反对数变换，用于恢复图像
flitered_image = uint8(flitered_image) %将图像转换为8位无符号整数格式
%图像归一化处理
%调整图像的对比度和亮度
flitered_image = imadjust(flitered_image)

figure
imshow(gray_image)
title('灰度图')
figure
imshow(average_image)
title('直方图均衡化图')
figure
imshow(flitered_image)
title('同态滤波图')
```

2.2. 运行结果

直方图均衡化图

同态滤波图

2.3. 两种算法效果分析

2.3.1. 视觉效果

直方图均衡化效果明显不如同态滤波法的效果，直方图均衡化后的图像出现像素化，图片明显不连续，而同态滤波效果视觉效果更好。

2.3.2. 性能指标

主要从对比度、图像亮度、均方误差 MSE、结构相似性 SSIM、峰值信噪比 PSNR、边缘强度方面来评估两种方法的效果，具体代码如下：

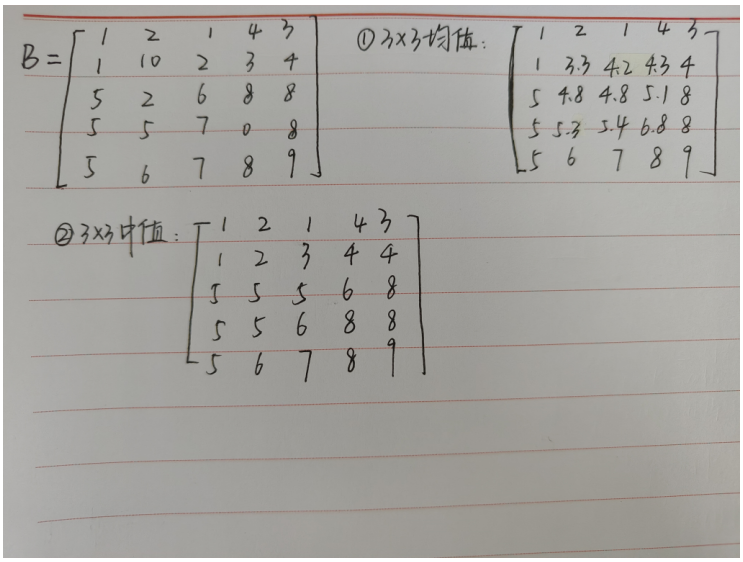
```
% 计算对比度
contrast_original = std(double(gray_image(:))); % 原始图像的对比度
contrast_average = std(double(average_image(:))); % 均衡化图像的对比度
contrast_filtered = std(double(filtered_image(:))); % 同态滤波图像的对比度

% 计算亮度
brightness_original = mean(double(gray_image(:))); % 原始图像的亮度
brightness_average = mean(double(average_image(:))); % 均衡化图像的亮度
brightness_filtered = mean(double(filtered_image(:))); % 同态滤波图像的亮度

% 计算均方误差（MSE）
mse_average = immse(average_image, gray_image); % 均衡化与原始图像的 MSE
mse_filtered = immse(filtered_image, gray_image); % 同态滤波与原始图像的 MSE

% 计算 SSIM 结构相似性指数，越接近1寿命图像之间的结构相似度越高
ssim_average = ssim(average_image, gray_image); % 均衡化与原始图像的 SSIM
ssim_filtered = ssim(filtered_image, gray_image); % 同态滤波与原始图像的 SSIM

% 计算 PSNR 峰值信噪比 越高表示图像质量越好
psnr_average = psnr(average_image, gray_image); % 均衡化图像的 PSNR
psnr_filtered = psnr(filtered_image, gray_image); % 同态滤波图像的 PSNR

% 计算边缘强度
edges_original = edge(gray_image, 'Canny'); % 原始图像的边缘
edges_average = edge(average_image, 'Canny'); % 均衡化图像的边缘
edges_filtered = edge(filtered_image, 'Canny'); % 同态滤波图像的边缘

% 计算边缘数量
edge_count_original = sum(edges_original(:));
edge_count_average = sum(edges_average(:));
edge_count_filtered = sum(edges_filtered(:));
```

```
% 显示结果
fprintf('原始图像对比度: %.2f\n', contrast_original);
fprintf('均衡化图像对比度: %.2f\n', contrast_average);
fprintf('同态滤波图像对比度: %.2f\n', contrast_filtered);

fprintf('原始图像亮度: %.2f\n', brightness_original);
fprintf('均衡化图像亮度: %.2f\n', brightness_average);
fprintf('同态滤波图像亮度: %.2f\n', brightness_filtered);

fprintf('均衡化 MSE: %.2f\n', mse_average);
fprintf('同态滤波 MSE: %.2f\n', mse_filtered);

fprintf('均衡化 SSIM: %.2f\n', ssim_average);
fprintf('同态滤波 SSIM: %.2f\n', ssim_filtered);

fprintf('均衡化 PSNR: %.2f dB\n', psnr_average);
fprintf('同态滤波 PSNR: %.2f dB\n', psnr_filtered);

fprintf('原始图像边缘数量: %d\n', edge_count_original);
fprintf('均衡化图像边缘数量: %d\n', edge_count_average);
fprintf('同态滤波图像边缘数量: %d\n', edge_count_filtered);
```

运行结果如下：

```
原始图像对比度: 67.14
均衡化图像对比度: 73.98
同态滤波图像对比度: 74.53
原始图像亮度: 35.62
均衡化图像亮度: 127.59
同态滤波图像亮度: 38.33
均衡化 MSE: 11311.74
同态滤波 MSE: 63.71
均衡化 SSIM: 0.18
同态滤波 SSIM: 0.95
均衡化 PSNR: 7.60 dB
同态滤波 PSNR: 30.09 dB
原始图像边缘数量: 22547
均衡化图像边缘数量: 64258
同态滤波图像边缘数量: 22602
```

分析可知：均衡化大幅提高了亮度，但是均方误差很大，与原图之间的结构相似性也很低，信噪比偏低说明图像质量很一般，边缘数量虽然多但是与原图无关；同态滤波保留了原图的相关细节，结构相似性很高，均方误差也更小，小幅提升了对比度和亮度，图像质量较好。


B=
1 2 1 4 3
1 10 2 3 4
5 2 6 8 8
5 5 7 0 8
5 6 7 8 9
①3×3均值:
1 2 1 4 3
1 3.3 4.2 4.3 4
5 4.8 4.8 5.1 8
5 5.3 5.4 6.8 8
5 6 7 8 9
②3×3中值:
1 2 1 4 3
1 2 3 4 4
5 5 5 6 8
5 5 6 8 8
5 6 7 8 9