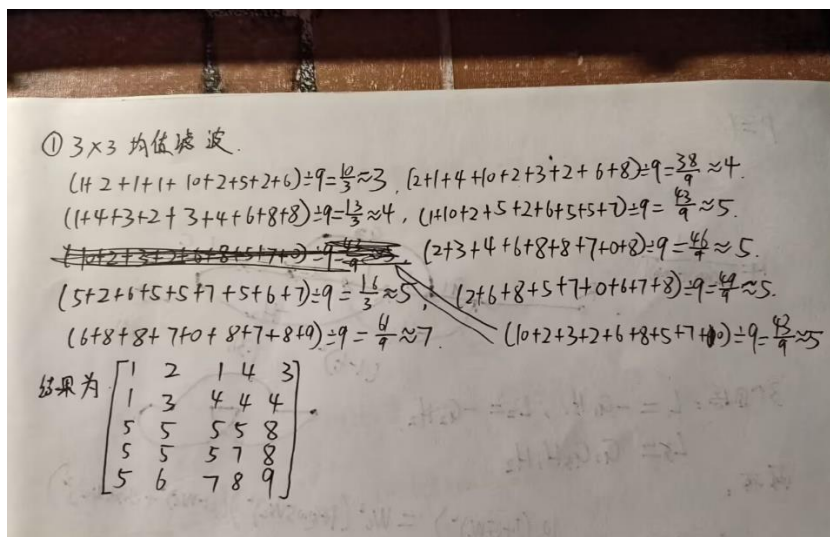


题目①

1. 处理步骤

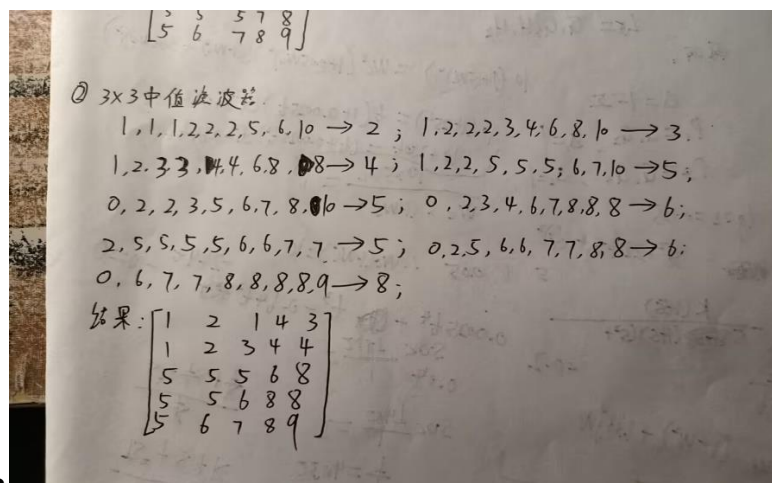
均值滤波:

- 首先, 确定原始图像。
- 对于图像内部像素 (除边框外), 取其 3×3 邻域, 计算邻域内像素值的平均值, 以此平均值作为该像素滤波后的新值。
- 保持图像边框像素不变, 即滤波后的图像边框与原始图像边框相同。



中值滤波:

- 同样先明确原始图像。
- 针对图像内部像素, 获取其 3×3 邻域, 算出邻域内像素值的中值, 将中值作为该像素滤波后的新值。
- 如同均值滤波, 保留图像边框像素与原始图像一致。



2. 代码

```
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];
```

```
% 调用均值滤波函数并显示结果
```

```
filtered_image_mean = mean_filter(B);  
disp('均值滤波后的图像: ');  
disp(filtered_image_mean);
```

```
% 调用中值滤波函数并显示结果
```

```
filtered_image_median = median_filter(B);  
disp('中值滤波后的图像: ');  
disp(filtered_image_median);
```

```
% 均值滤波函数
```

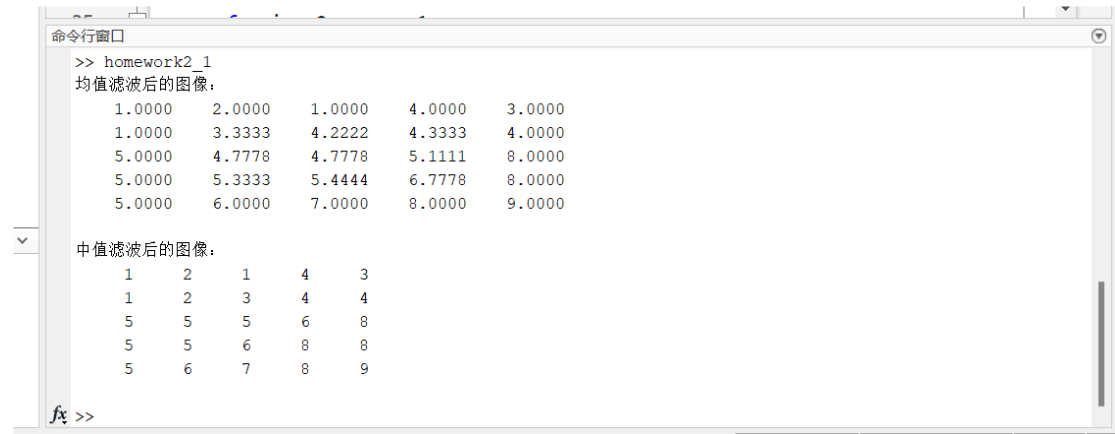
```
function filtered_image = mean_filter(image)  
B = [1 2 1 4 3;1 10 2 3 4;5 2 6 8 8;5 5 7 0 8;5 6 7 8 9];  
[rows, cols] = size(image);  
filtered_image = zeros(rows, cols);  
for i = 2:rows - 1  
    for j = 2:cols - 1  
        neighborhood = image(i - 1:i + 1, j - 1:j + 1);  
        filtered_image(i, j) = mean(neighborhood(:));  
    end  
end  
filtered_image(1,:) = B(1,:);  
filtered_image(end,:) = B(end,:);  
filtered_image(:,1) = B(:,1);  
filtered_image(:,end) = B(:,end);  
end
```

```
% 中值滤波函数
```

```
function filtered_image = median_filter(image)  
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];  
[rows, cols] = size(image);  
filtered_image = zeros(rows, cols);  
for i = 2:rows - 1  
    for j = 2:cols - 1  
        neighborhood = image(i - 1:i + 1, j - 1:j + 1);  
        filtered_image(i, j) = median(neighborhood(:));  
    end  
end  
filtered_image(1,:) = B(1,:);  
filtered_image(end,:) = B(end,:);
```

```
filtered_image(:,1) = B(:,1);  
filtered_image(:,end) = B(:,end);  
end
```

3. 运行结果



命令窗口

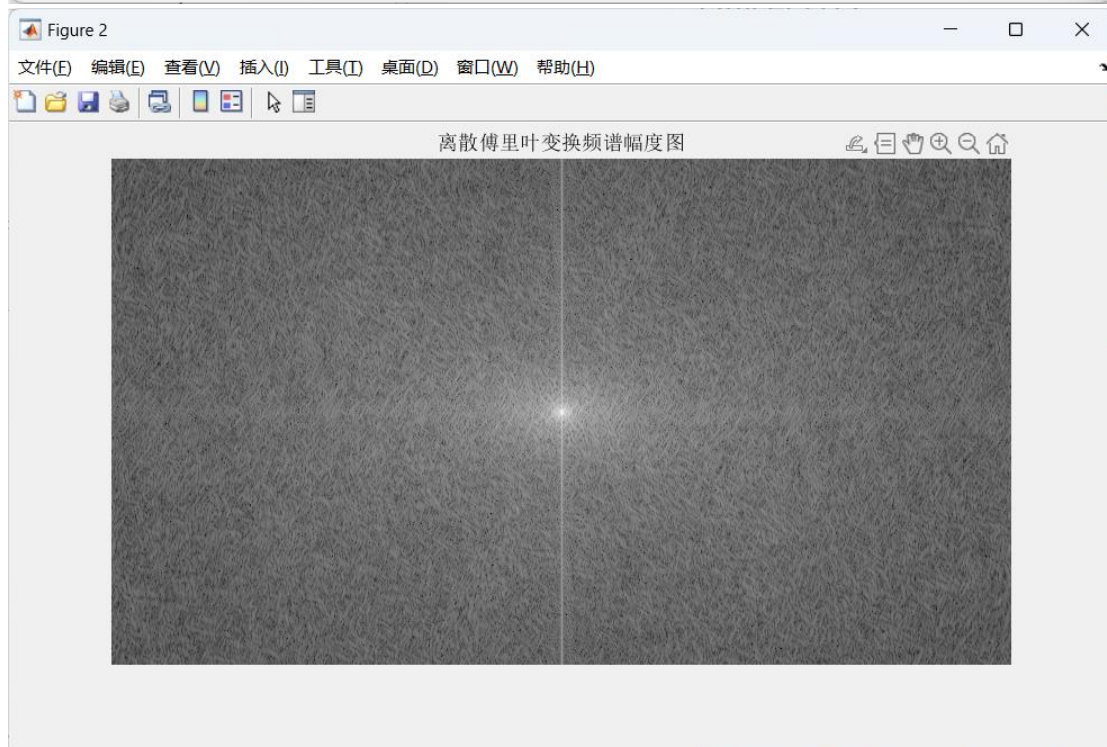
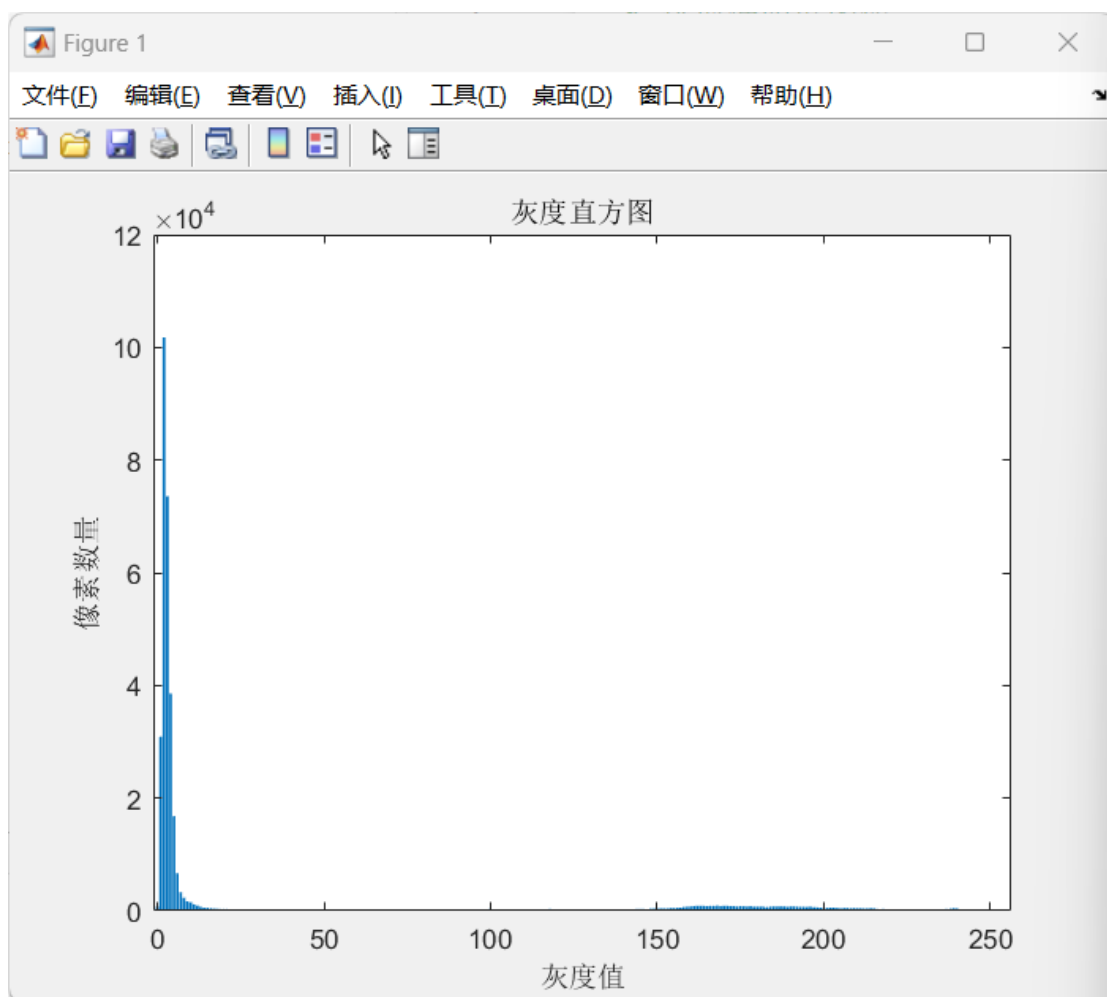
```
>> homework2_1  
均值滤波后的图像:  
1.0000    2.0000    1.0000    4.0000    3.0000  
1.0000    3.3333    4.2222    4.3333    4.0000  
5.0000    4.7778    4.7778    5.1111    8.0000  
5.0000    5.3333    5.4444    6.7778    8.0000  
5.0000    6.0000    7.0000    8.0000    9.0000  
  
中值滤波后的图像:  
1     2     1     4     3  
1     2     3     4     4  
5     5     5     6     8  
5     5     6     8     8  
5     6     7     8     9
```

fx >>

题目②

一、 对上述低照度图像进行灰度化，计算并显示以上低照度图像的灰度直方图和离散傅里叶变换频谱幅度图；

a) 运行结果：



b) 代码:

```
2. % 读取低照度图像
3. img = imread('低照度图像.jpg');
4.
5. % 图像灰度化
6. gray_img = rgb2gray(img);
7.
8. % 计算并显示灰度直方图
9. hist = imhist(gray_img);
10. figure;
11. bar(0:255, hist);
12. title('灰度直方图');
13. xlabel('灰度值');
14. ylabel('像素数量');
15.
16. % 计算离散傅里叶变换频谱幅度图
17. f = fft2(gray_img);
18. fshift = fftshift(f);
19. magnitude_spectrum = abs(fshift);
20. figure;
21. imshow(log(magnitude_spectrum + 1), []);
22. title('离散傅里叶变换频谱幅度图');
```

二、 对以上低照度图像分别进行直方图均衡化和同态滤波操作，
并对两种算法的最终效果进行对比；

a) 运行结果:



b) 代码:

```
% 读取低照度图像
img = imread('低照度图像.jpg');
% 图像灰度化
gray_img = rgb2gray(img);

% 计算图像的直方图
histogram = zeros(1, 256);% 创建一个长度为 256 的全零向量 histogram，用于存储图
像的直方图。
[rows, cols] = size(gray_img);% 获取灰度图像 gray_img 的行数 rows 和列数
cols。
for i = 1:rows
    for j = 1:cols
        pixel_value = gray_img(i, j);% 获取当前像素的灰度值 pixel_value。
        histogram(pixel_value + 1) = histogram(pixel_value + 1) + 1;% 将
        histogram 中对应灰度值的位置加 1，因为灰度值范围是 0-255，所以 pixel_value + 1
        作为索引。
    end
end

% 计算累积分布函数 (CDF)
cdf = zeros(1, 256);% 创建一个长度为 256 的全零向量 cdf，用于存储累积分布函数。
cdf(1) = histogram(1);% 初始化 cdf 的第一个值为 histogram 的第一个值。
for i = 2:256
    cdf(i) = cdf(i - 1) + histogram(i);% 通过循环计算累积分布函数，当前位置的
    cdf 值等于前一个位置的值加上当前灰度值对应的直方图值。
end

% 归一化 CDF
cdf_normalized = cdf / (rows * cols);
% 将累积分布函数 cdf 进行归一化，除以图像的总像素数 rows * cols，使得结果在 0 到
1 之间。

% 进行直方图均衡化
equalized_img = zeros(rows, cols, 'like', gray_img);
for i = 1:rows
    for j = 1:cols
        pixel_value = gray_img(i, j);
        equalized_img(i, j) = round(cdf_normalized(pixel_value + 1) * 255);
    end
end

% 同态滤波
```

```

% 首先进行对数变换
log_img = log(double(gray_img)+1);
% 对灰度图像进行对数变换，目的是将图像的乘法运算转换为加法运算，
% 便于后续在频域中分别处理照度和反射率分量。加 1 是为了避免对 0 值取对数。

% 进行二维离散傅里叶变换
f = fft2(log_img);
% 对对数变换后的图像进行二维离散傅里叶变换，将图像从空间域转换到频域。

% 设定滤波器参数
M = size(log_img,1); % 获取图像的行数
N = size(log_img,2); % 获取图像的列数
u = 0:(M-1); % 创建一个从 0 到 M-1 的向量，表示图像的行索引
v = 0:(N-1); % 创建一个从 0 到 N-1 的向量，表示图像的列索引
idx = find(u>M/2); % 找到大于图像行数一半的索引
u(idx) = u(idx)-M; % 对于大于行数一半的索引，将其减去行数，实现中心对称
idy = find(v>N/2); % 找到大于图像列数一半的索引
v(idy) = v(idy)-N; % 对于大于列数一半的索引，将其减去列数，实现中心对称
[V,U] = meshgrid(v,u); % 创建二维网格，用于后续计算频率坐标
D = sqrt(U.^2 + V.^2); % 计算频率坐标的模，即到频率中心的距离
H = (0.5 + 2 * ((D./(D + 0.5)))); % 定义同态滤波器函数，用于调整照度和反射率分量

% 滤波
filtered_f = H.* f;
% 将滤波器与频域图像相乘，实现滤波操作。

% 逆傅里叶变换和指数变换
filtered_img = real(exp(iff2(filtered_f)) - 1);
% 对滤波后的图像进行逆傅里叶变换，将其从频域转换回空间域。
% 然后进行指数变换，恢复到原始图像的形式，再减 1 与前面的对数变换对应。

% 显示直方图均衡化图像和同态滤波图像
figure;
subplot(1,2,1);
imshow(equalized_img);
title('直方图均衡化图像');
subplot(1,2,2);
imshow(filtered_img);
title('同态滤波图像');

```