注家豪

一、

均值滤波：

$$\begin{bmatrix} 1 & 2 & 1 & 4 & 3 \\ 1 & 10 & 2 & 3 & 4 \\ 5 & 2 & 6 & 8 & 8 \\ 5 & 5 & 7 & 0 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

滤波器：$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

保留边框元素，滤波结果为

$$\begin{bmatrix} 1 & 2 & 1 & 4 & 3 \\ 1 & 3.33 & 4.22 & 4.33 & 4 \\ 5 & 4.78 & 4.77 & 5.11 & 8 \\ 5 & 5.33 & 5.44 & 6.78 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

中值滤波：

在3×3范围内找到中值

$\begin{matrix} 1 & 2 & 1 \\ 1 & 10 & 2 \\ 5 & 2 & 6 \end{matrix}$ 中值为2，$\begin{matrix} 2 & 1 & 4 \\ 10 & 2 & 3 \\ 2 & 6 & 8 \end{matrix}$ 中值为3，$\begin{matrix} 1 & 4 & 3 \\ 3 & 4 & 4 \\ 5 & 6 & 8 \end{matrix}$ 中值为4.

...

结果为：

$$\begin{bmatrix} 1 & 2 & 1 & 4 & 3 \\ 1 & 2 & 3 & 4 & 4 \\ 5 & 5 & 5 & 6 & 8 \\ 5 & 5 & 6 & 8 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

二、

转为灰度图：

原图

(1) 计算并显示以上低照度图像的灰度直方图

原理：遍历图像中的每个像素，统计每个灰度值的出现次数。

分析：

初始化一个大小为 256 的数组 huidu_zhifangtu，用于存储每个灰度级别（0-255）的像素数量；

使用双重循环遍历每个像素，通过 pixel_value+1 将灰度值的计数增加 1。

```
% 计算灰度直方图
function huidu_zhifangtu=count_huidu(img)
% 获取图像的大小
[rows,cols]=size(img);

% 初始化灰度直方图数组
huidu_zhifangtu=zeros(1,256);

% 遍历图像，统计各灰度的像素个数
for i=1:rows
    for j=1:cols
        pixel_value=img(i,j);
        huidu_zhifangtu(pixel_value+1)=huidu_zhifangtu(pixel_value+1)+1;
    end
end
end
```

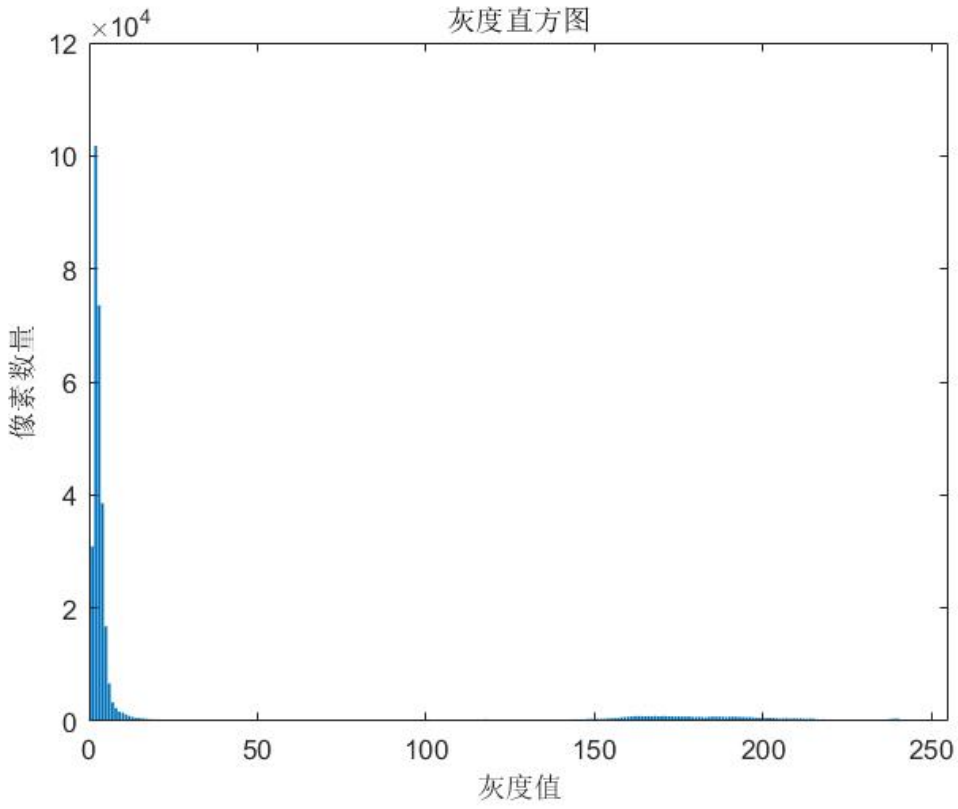
之后展示计算结果

```
zhifangtu=count_huidu(img);

% 绘制灰度直方图
figure;
bar(0:255, zhifangtu);
```

```
title('灰度直方图');
xlabel('灰度值');
ylabel('像素数量');
xlim([0 255]);
```

结果：



(2) 计算离散傅里叶变换频谱幅度图

原理：将图像从空间域转换到频域，能够分析图像中的频率成分。通

一维离散傅里叶变换

对于一个长度为 N 的序列 $x[n]$，其DFT $X[k]$ 定义为：

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-2\pi i \frac{kn}{N}}, \quad k = 0, 1, \ldots, N-1$$

二维离散傅里叶变换

对于一个大小为 $M \times N$ 的二维信号 $f(m,n)$，其DFT $F(u,v)$ 定义为：

$$F(u,v) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) \cdot e^{-2\pi i \left(\frac{um}{M} + \frac{vn}{N}\right)}, \quad u = 0, 1, \ldots, M-1, \quad v = 0, 1, \ldots, N-1$$

由于直接对图像进行二维傅里叶变换非常耗时，因此通过对图像的行和列分别进行一维 DFT，从而实现二维傅里叶变换。

$$F(x,v)=\frac{1}{N}\sum_{y=0}^{N-1}f(x,y)\exp[-j2\pi vy/N]$$

$$F(u,v)=\frac{1}{N}\sum_{x=0}^{N-1}F(x,v)\exp[-j2\pi ux/N]$$

分析：首先读取图像并转换为灰度图，同 2，接着调用手动实现的二维傅里叶变换 my_dft2(img)，第一部分循环遍历每一行，使用 my_dft1d 函数计算一维傅里叶变换，第二部分循环遍历每一列，计算结果矩阵 F 的列的一维傅里叶变换：

```
function F = my_dft2(img)
[M, N] = size(img);
F = zeros(M, N);
% 对每一行进行 DFT
for m = 1:M
    F(m, :) = my_dft1d(img(m, :));
End

% 对每一列进行 DFT
for n = 1:N
    F(:, n) = my_dft1d(F(:, n)); % 对每一列进行一维傅里叶变换
end
end
```

上述代码中的一维傅里叶变换函数 my_dft1d(x)，：

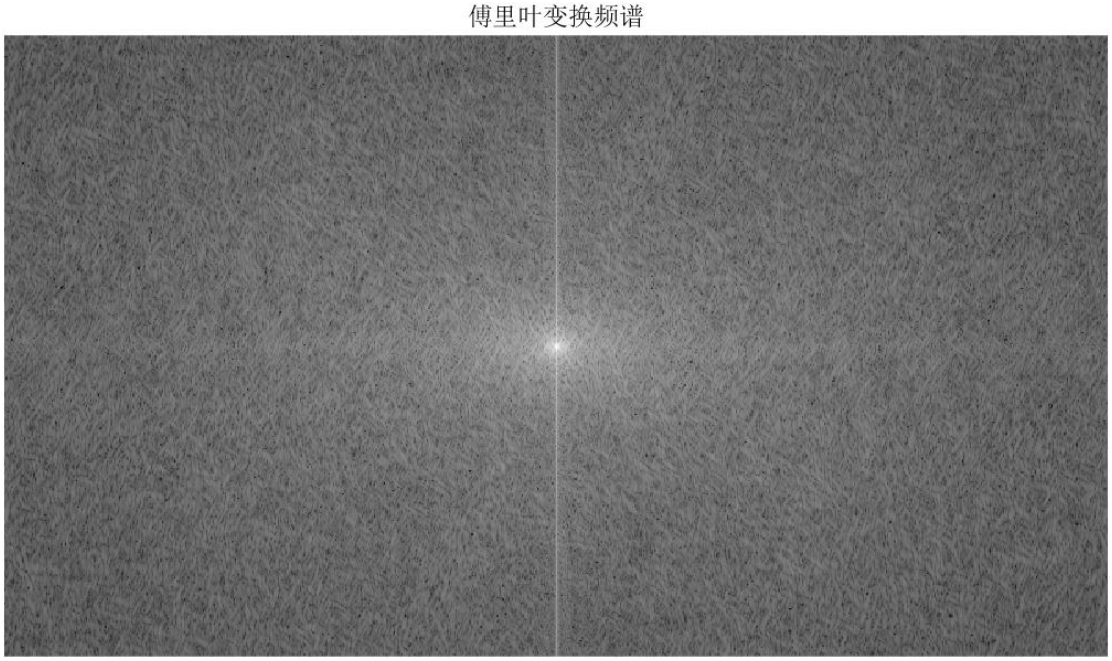
```
function X = my_dft1d(x)
N = length(x);
X = zeros(1, N);
% 计算傅里叶变换
for k = 1:N
    for n = 1:N
        X(k) = X(k) + x(n) * exp(-2 * pi * 1i * (k-1) * (n-1) / N);
    end
end
end
```

调用后还需将频谱的低频成分移到中心，并可视化结果

```
% 计算二维 DFT
aaa=my_dft2(double(img));
% 计算幅度谱并取对数
magnitude_spectrum_manual = log(1 + abs(fftshift(aaa)));
figure;
```

```
imshow(magnitude_spectrum_manual, []);
title("傅里叶变换频谱");
```

结果：



(3) 对原灰度图进行直方图均衡化

原理：将原图像的灰度值重新映射，使得像素值分布更加均匀。这可以有效地增强低对比度图像，使得图像的细节更加突出。

累计分布函数 (CDF)：用于重新分配灰度级，公式如下：

$$s_k = \frac{(L-1)}{MN} \sum_{j=0}^{k} h(j)$$

其中：

- s_k 为均衡化后的灰度值。
- L 为灰度级总数 (256)。
- M 和 N 分别是图像的高度和宽度。
- $h(j)$ 是灰度级 j 的像素数量。

分析：

使用 cumsum 函数计算灰度直方图的累计分布函数 (CDF)；

将 CDF 归一化到 [0,1] 范围，然后通过线性映射调整到 [0,255] 范围；

遍历图像像素，将原灰度值替换为均衡化后的灰度值，得到增强后的图像。

```
% 均衡化直方图
function equalized_img=equalized(img,zhifangtu)
```

```
[rows,cols]=size(img);
% 计算累计分布函数
cdf = cumsum(zhifangtu); % 累加直方图值
cdf_normalized = cdf / numel(img); % 将 CDF 归一化到[0,1]之间

% 构建均衡化后的灰度值映射表
% 将归一化的 CDF 映射到[0, 255]之间
equalized_mapping = uint8(255 * cdf_normalized);

% 应用均衡化映射表
equalized_img = zeros(size(img), 'uint8');
for i = 1:rows
    for j = 1:cols
        equalized_img(i, j) = equalized_mapping(img(i, j) + 1); % 应用灰度值
        映射
    end
end

end
```

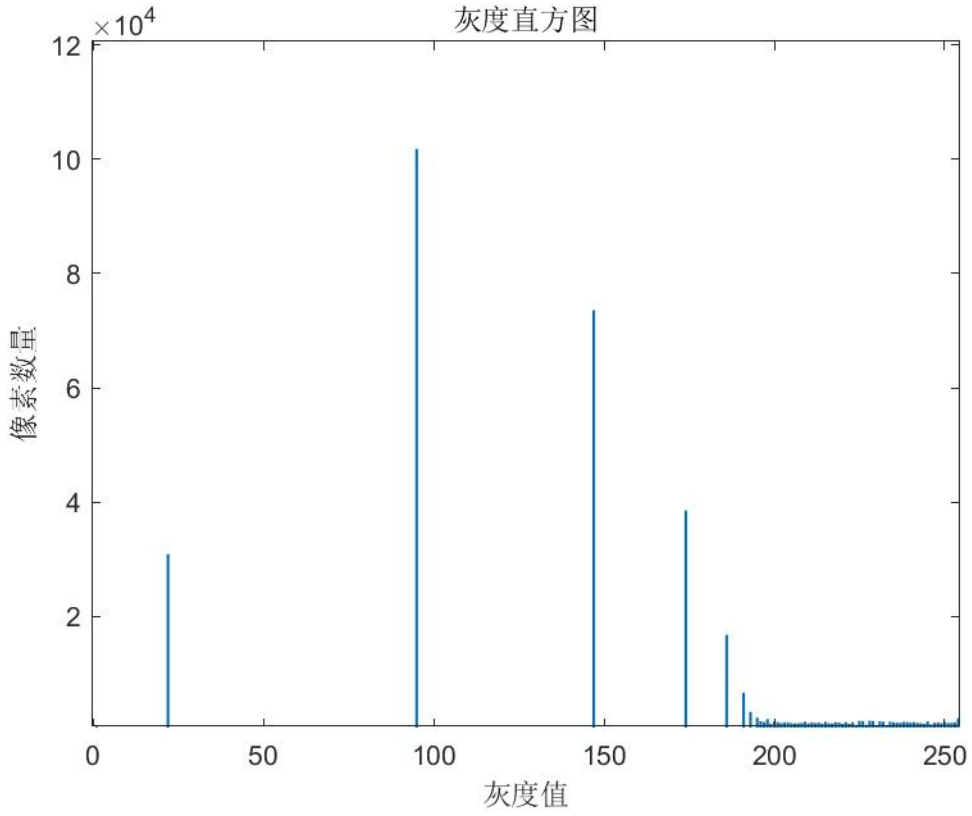
调用该函数，绘制直方图均衡化后的图像，计算其灰度直方图并可视化

```
% 绘制直方图均衡化后的图像
figure;
imshow(equalized_img);
title('直方图均衡化后的图像');

equalized_zhifangtu=count_huidu(equalized_img);
% 绘制灰度直方图
figure;
bar(0:255, equalized_zhifangtu);
title('灰度直方图');
xlabel('灰度值');
ylabel('像素数量');
xlim([0 255]);
```

结果：

直方图均衡化后的图像



(4) 对原灰度图进行同态滤波

原理：同态滤波是一种在频域中同时进行图像对比度增强和压缩图像亮度范围的滤波方法；其基本思想是减少入射分量 i(x,y)，并同时增加反射分量 r(x,y)来改善图像 f(x,y)的显示效果；i(x,y)在空间上变化缓慢，其频谱集中在低频段，r(x,y)反映图像的细节和边缘，其频谱集中在高频段。

分析：

对数变换：将图像转换到对数域中。因为在对数域中，图像的乘法关系可以转变为加法关系，这使得低频和高频信息分离变得更简单。将图像转换为 double 类型，并在每个像素上加 1。log 函数将图像进行对数变换，得到 log_img。

```
% 转换图像为浮点型，并进行对数变换
img = double(img) + 1; % 避免对数运算中的零值
log_img = log(img); % 对数变换
```

使用自定义的 my_dft2 函数对 log_img 进行二维傅里叶变换，得到频域表示 F。

```
F = my_dft2(log_img); % 计算二维离散傅里叶变换
```

为滤波器设计一个频率网格 D，D 表示距离图像中心的频率距离。傅里叶变换的结果集中低频成分在中心，高频成分在外围。通过 u - cols/2 和 v - rows/2 进行移位，使得频域图像的中心对齐到图像中心点。

meshgrid 函数生成图像的频率坐标网格。使用 sqrt 计算每个坐标到图像中心的距离，得到频率距离 D。

```
% 创建频率域的网格
[u, v] = meshgrid(0:cols-1, 0:rows-1);
D = sqrt((u - cols/2).^2 + (v - rows/2).^2); % 频率距离
```

创建同态滤波器：exp(-(D.^2) / (2 * (cutoff^2))) 表示高斯函数的形式，用于定义滤波器形状。通过 gamma_h 和 gamma_l 的线性插值来调节滤波器的响应。

```
% 创建同态滤波器
H = 1 - exp(-(D.^2) / (2 * (cutoff^2))); % 理想低通滤波器
H = gamma_l + (gamma_h - gamma_l) * H; % 线性插值
```

频域滤波

```
G = H .* F; % 应用同态滤波器
```

逆傅里叶变换：恢复图像。通过 my_idft2 函数将频域中的图像转换回空间域。

逆对数变换：之前对图像进行了对数变换，现在通过 exp 操作将其恢复。

```
% 计算逆傅里叶变换
filtered_log_img = my_idft2(G); % 计算逆 2D DFT

% 对数变换的逆变换
filtered_img = exp(filtered_log_img) - 1; % 逆对数变换
filtered_img = uint8(filtered_img); % 转换为 uint8 类型

% 逆二维离散傅里叶变换函数
function img = my_idft2(F)
[M, N] = size(F);
img = zeros(M, N);
% 对每一列进行逆 DFT
for n = 1:N
    img(:, n) = my_idft1d(F(:, n));
```

```
end

% 对每一行进行逆 DFT
for m = 1:M
    img(m, :) = my_idft1d(img(m, :));
end
img = img / (M * N); % 归一化

end

% 逆一维离散傅里叶变换
function x = my_idft1d(X)
N = length(X); % 输入向量的长度
x = zeros(1, N); % 初始化输出向量

% 一维 IDFT 计算
for n = 1:N
    for k = 1:N
        x(n) = x(n) + X(k) * exp(2 * pi * 1i * (k-1) * (n-1) / N); % 逆傅里叶
        变换公式
    end
end

end
```

限制像素值范围

```
% 限制图像值在 [0, 255] 之间
filtered_img(filtered_img > 255) = 255;
filtered_img(filtered_img < 0) = 0;
```

调用该函数，并展示结果。

```
filtered_img = homomorphic_filter(img, 30, 2.7, 0.5);
figure;
imshow(filtered_img);
title('同态滤波后的图像');
```

结果：

同态滤波后的图像

通过对比，可以得出对于低照度图像，同态滤波通常效果更好。
原因：
同态滤波能够将图像分解为光照（低频）和细节（高频）分量，并分别处理。因此，可以抑制光照不均匀的影响，同时增强细节。
直方图均衡化则只是单纯地重新分配灰度值，不能区分光照和细节的不同作用，容易导致图像看起来过度增强且不自然。

同态滤波能够根据频率特性来控制细节的增强（通过增益参数 gamma_h 和 gamma_l），使得高频细节更加突出，但不会影响整体亮度。这对于在低光条件下突出重要细节非常有用。
直方图均衡化只能整体提升对比度，不能精确地控制细节增强效果。

同态滤波的频率滤波特性使得它可以平滑大范围的光照变化，从而减少因光照不均匀引起的亮度不均匀。
直方图均衡化没有这个特性，对于有明显光照变化的图像，可能会使得某些区域变得过亮或过暗。

代码：

```
clc,clear;
% 读取图像并转换为灰度图像
img=imread('低照度图像.jpg');
img=rgb2gray(img); % 转换为灰度图像
[rows,cols]=size(img); % 获取图像的高和宽

% 绘制原图
figure;
imshow(img);
title('原图');

zhifangtu=count_huidu(img);

% 绘制灰度直方图
figure;
bar(0:255, zhifangtu);
title('灰度直方图');
xlabel('灰度值');
ylabel('像素数量');
xlim([0 255]);


equalized_img=equalized(img,zhifangtu);

% 绘制直方图均衡化后的图像
figure;
imshow(equalized_img);
title('直方图均衡化后的图像');

equalized_zhifangtu=count_huidu(equalized_img);
% 绘制灰度直方图
figure;
bar(0:255, equalized_zhifangtu);
title('灰度直方图');
xlabel('灰度值');
ylabel('像素数量');
xlim([0 255]);

% 计算二维 DFT
aaa=my_dft2(double(img));
% 计算幅度谱并取对数
magnitude_spectrum_manual = log(1 + abs(fftshift(aaa)));
figure;
```

```
imshow(magnitude_spectrum_manual, []);
title("傅里叶变换频谱");

filtered_img = homomorphic_filter(img, 30, 2.7, 0.5);
figure;
imshow(filtered_img);
title('同态滤波后的图像');

% 计算灰度直方图
function huidu_zhifangtu=count_huidu(img)
% 获取图像的大小
[rows,cols]=size(img);

% 初始化灰度直方图数组
huidu_zhifangtu=zeros(1,256);

% 遍历图像，统计各灰度的像素个数
for i=1:rows
for j=1:cols
pixel_value=img(i,j);
huidu_zhifangtu(pixel_value+1)=huidu_zhifangtu(pixel_value+1)+1;
end
end
end

% 均衡化直方图
function equalized_img=equalized(img,zhifangtu)
[rows,cols]=size(img);
% 计算累计分布函数
cdf = cumsum(zhifangtu); % 累加直方图值
cdf_normalized = cdf / numel(img); % 将 CDF 归一化到[0,1]之间

% 构建均衡化后的灰度值映射表
% 将归一化的 CDF 映射到[0, 255]之间
equalized_mapping = uint8(255 * cdf_normalized);

% 应用均衡化映射表
equalized_img = zeros(size(img), 'uint8');
for i = 1:rows
for j = 1:cols
equalized_img(i, j) = equalized_mapping(img(i, j) + 1); % 应用灰度值映射
end
end
```

```
end

% 二维离散傅里叶变换
function F = my_dft2(img)
[M, N] = size(img);

F = zeros(M, N);
% 对每一行进行 DFT
for m = 1:M
F(m, :) = my_dft1d(img(m, :));
end

% 对每一列进行 DFT
for n = 1:N
F(:, n) = my_dft1d(F(:, n));
end
end

% 实现一维离散傅里叶变换
function X = my_dft1d(x)
N = length(x);
X = zeros(1, N);
% 一维傅里叶变换
for k = 1:N
for n = 1:N
X(k) = X(k) + x(n) * exp(-2 * pi * 1i * (k-1) * (n-1) / N);
end
end
end

% 同态滤波函数
% 输入：
% img: 输入的灰度图像
% cutoff: 截止频率
% gamma_h: 高频成分增益
% gamma_l: 低频成分增益
function filtered_img = homomorphic_filter(img, cutoff, gamma_h, gamma_l)
% 转换图像为浮点型，并进行对数变换
img = double(img) + 1; % 避免对数运算中的零值
log_img = log(img); % 对数变换

% 获取图像的大小
[rows, cols] = size(log_img);
```

```
% 创建二维傅里叶变换
F = my_dft2(log_img); % 计算二维离散傅里叶变换

% 创建频率域的网格
[u, v] = meshgrid(0:cols-1, 0:rows-1);
D = sqrt((u - cols/2).^2 + (v - rows/2).^2); % 频率距离

% 创建同态滤波器
H = 1 - exp(-(D.^2) / (2 * (cutoff^2))); % 理想低通滤波器
H = gamma_l + (gamma_h - gamma_l) * H; % 线性插值

% 频域乘法
G = H .* F; % 应用同态滤波器

% 计算逆傅里叶变换
filtered_log_img = my_idft2(G); % 计算逆 2D DFT

% 对数变换的逆变换
filtered_img = exp(filtered_log_img) - 1; % 逆对数变换
filtered_img = uint8(filtered_img); % 转换为 uint8 类型

% 限制图像值在 [0, 255] 之间
filtered_img(filtered_img > 255) = 255;
filtered_img(filtered_img < 0) = 0;
end


% 逆二维离散傅里叶变换函数
function img = my_idft2(F)
[M, N] = size(F);
img = zeros(M, N);
% 对每一列进行逆 DFT
for n = 1:N
img(:, n) = my_idft1d(F(:, n));
end

% 对每一行进行逆 DFT
for m = 1:M
img(m, :) = my_idft1d(img(m, :));
end
img = img / (M * N); % 归一化
end

% 逆一维离散傅里叶变换
```

```
function x = my_idft1d(X)
N = length(X); % 输入向量的长度
x = zeros(1, N); % 初始化输出向量

% 一维 IDFT 计算
for n = 1:N
for k = 1:N
x(n) = x(n) + X(k) * exp(2 * pi * 1i * (k-1) * (n-1) / N); % 逆傅里叶变换公式
end
end
end
```