傅里叶变换和傅里叶系数主要用于将图像从空间域（即像素值）转换到频率域，以便进行滤波、压缩等处理：

1. 傅里叶变换的物理含义

从空间域到频率域的转换: 图像通常是空间域的表示，即像素的亮度值在二维平面上的分布。傅里叶变换将这个空间域的图像表示转换为频率域，其中每个点表示图像中的某一频率分量。

频率: 在频率域中，低频分量代表图像中大的、缓慢变化的结构（如平滑区域），而高频分量则表示细节、边缘、噪声等快速变化的部分。

幅度与相位:

幅度表示在图像中某个特定频率分量的强度。大的幅度对应于图像中这一频率分量占据较大的比例，可能是主要的结构或模式。

相位提供了频率分量在空间域中如何分布的信息，影响了图像的结构和形状。

2. 傅里叶系数的物理含义

频率分量的权重: 傅里叶系数是频率域的数值表示，它们描述了不同频率成分在原始图像中出现的权重。一个傅里叶系数代表图像中某个频率的振幅和相位信息。

实部与虚部: 傅里叶系数通常是复数，实部和虚部分别表示频率分量的余弦和正弦分量，它们共同定义了该频率分量的振幅和相位。

能量分布: 傅里叶系数的大小可以反映图像的能量分布，尤其是在某些频率范围内。较大的系数意味着该频率在图像中占主导地位。

3. 在图像处理中的应用

滤波: 通过选择性地保留或去除某些频率分量（如低通滤波器只保留低频分量，高通滤波器只保留高频分量），可以实现平滑、去噪或边缘增强等效果。

图像压缩: 在频率域中，许多图像的高频分量（尤其是细微细节或噪声）可以忽略，从而通过只存储主要的低频分量来压缩图像。

图像重建: 通过傅里叶逆变换，可以从频率域的数据重建出原始的空间域图像。重建的图像质量取决于保留的频率分量。

编写程序（建议 Matlab）对以上图像（自行转换为灰度图）展开（1）顺时针旋转 30 度；（2）基于最近邻和双线性插值将图像分别放大 2 倍和 4 倍，并形成实验报告

1.实验目的

通过手动实现顺时针旋转和图像缩放操作，掌握最近邻插值和双线性插值的原理和应用。

2.代码

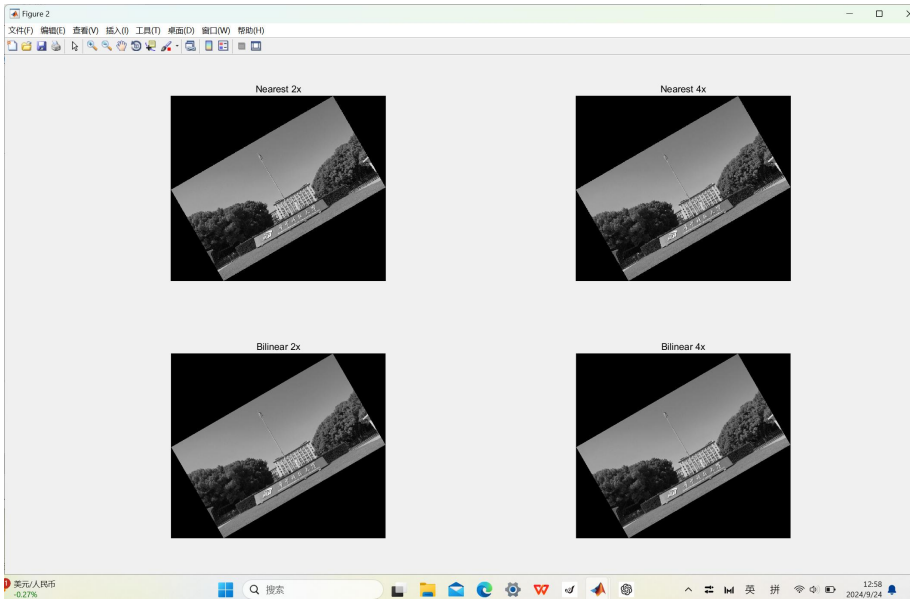
```
% 读取图像
img = imread('D:\picture.bmp');


% 转换为灰度图
gray_img = rgb2gray(img);

% 1. 顺时针旋转 30 度
angle = 30;   % 旋转角度
rotated_img = rotate_image(gray_img, angle);
```

```
% 2. 使用最近邻插值放大图像 2 倍和 4 倍
nearest_2x = nearest_neighbor_resize(rotated_img, 2);
nearest_4x = nearest_neighbor_resize(rotated_img, 4);

% 3. 使用双线性插值放大图像 2 倍和 4 倍
bilinear_2x = bilinear_resize(rotated_img, 2);
bilinear_4x = bilinear_resize(rotated_img, 4);

% 显示结果
figure;
subplot(2,2,1), imshow(nearest_2x), title('Nearest 2x');
subplot(2,2,2), imshow(nearest_4x), title('Nearest 4x');
subplot(2,2,3), imshow(bilinear_2x), title('Bilinear 2x');
subplot(2,2,4), imshow(bilinear_4x), title('Bilinear 4x');

% --------- 旋转函数 ----------
function rotated_img = rotate_image(img, angle)
    % 获取图像尺寸
    [rows, cols] = size(img);

    % 将角度转换为弧度
    rad = deg2rad(angle);

    % 计算旋转后的图像尺寸
    new_rows = round(abs(rows * cos(rad)) + abs(cols * sin(rad)));
    new_cols = round(abs(cols * cos(rad)) + abs(rows * sin(rad)));

    % 创建空白画布
    rotated_img = zeros(new_rows, new_cols, 'uint8');

    % 计算图像中心
    center_row = round(new_rows / 2);
    center_col = round(new_cols / 2);
    original_center_row = round(rows / 2);
    original_center_col = round(cols / 2);

    % 逐个像素映射
    for i = 1:new_rows
        for j = 1:new_cols
            % 将新图像的坐标映射到原始图像的坐标
            y = (i - center_row) * cos(rad) + (j - center_col) * sin(rad) + original_center_row;
            x = -(i - center_row) * sin(rad) + (j - center_col) * cos(rad) + original_center_col;
```

```
            % 检查坐标是否在原图范围内
            if (x >= 1 && x <= cols && y >= 1 && y <= rows)
                rotated_img(i,j) = img(round(y), round(x));
            end
        end
    end
end

% --------- 最近邻插值放大函数 ----------
function resized_img = nearest_neighbor_resize(img, scale)
    % 获取原图像尺寸
    [rows, cols] = size(img);

    % 计算新图像尺寸
    new_rows = round(rows * scale);
    new_cols = round(cols * scale);

    % 创建新图像
    resized_img = zeros(new_rows, new_cols, 'uint8');

    % 逐个像素赋值
    for i = 1:new_rows
        for j = 1:new_cols
            % 找到最近的原图像像素
            orig_i = round(i / scale);
            orig_j = round(j / scale);

            % 边界检查
            if orig_i < 1
                orig_i = 1;
            end
            if orig_j < 1
                orig_j = 1;
            end
            if orig_i > rows
                orig_i = rows;
            end
            if orig_j > cols
                orig_j = cols;
            end

            % 赋值
            resized_img(i,j) = img(orig_i, orig_j);
        end
```

```
    end
end

% ---------  双线性插值放大函数  ----------
%  修改  bilinear_resize  函数，确保下标是正整数
function resized_img = bilinear_resize(img, scale)
    [rows, cols] = size(img);
    new_rows = round(rows * scale);
    new_cols = round(cols * scale);

    resized_img = zeros(new_rows, new_cols, 'uint8');

    for i = 1:new_rows
        for j = 1:new_cols
            orig_x = (i - 0.5) / scale + 0.5;
            orig_y = (j - 0.5) / scale + 0.5;

            %  边界检查，确保下标为正整数且不超出原图尺寸
            x1 = max(1, floor(orig_x));
            x2 = min(rows, ceil(orig_x));
            y1 = max(1, floor(orig_y));
            y2 = min(cols, ceil(orig_y));

            dx = orig_x - x1;
            dy = orig_y - y1;

            value = (1 - dx) * (1 - dy) * double(img(x1, y1)) + ...
                    dx * (1 - dy) * double(img(x2, y1)) + ...
                    (1 - dx) * dy * double(img(x1, y2)) + ...
                    dx * dy * double(img(x2, y2));

            resized_img(i,j) = round(value);
        end
    end
end
```

3.实验结果



4.实验步骤

（1）图像读取与灰度化

将输入图像读取为灰度图。

（2）图像旋转

手动实现图像顺时针旋转 30 度，结果保存在 rotated_img 中。

（3）最近邻插值放大

使用手动实现的最近邻插值方法，将旋转后的图像分别放大 2 倍和 4 倍。

（4）双线性插值放大

使用手动实现的双线性插值方法，将旋转后的图像分别放大 2 倍和 4 倍。

5.实验结论

最近邻插值计算速度较快，但插值结果会出现较明显的块状效应。

双线性插值能够较好地平滑图像，但计算复杂度相对较高。

编写程序（建议 Matlab）对以上图像（自行转换为灰度图）展开傅里叶变换，提取傅里叶变换图像（将频率原点移至图像中心），并形成实验报告。

1.实验目的：

手动实现二维傅里叶变换与频谱移位，理解频域分析在图像处理中的应用。

2.实验步骤：

（1）图像读取与预处理：

读取原始图像并将其转换为灰度图像。

（2）二维傅里叶变换的实现：

使用手动实现的 2D DFT 函数对图像进行傅里叶变换，将其从空间域转换到频率域。

（3）频谱移位：

将频谱原点移至图像中心，以便更直观地观察频域特性。

（4）频域特性分析：

使用对数缩放显示傅里叶变换后的频谱图，并观察高频与低频区域的分布。

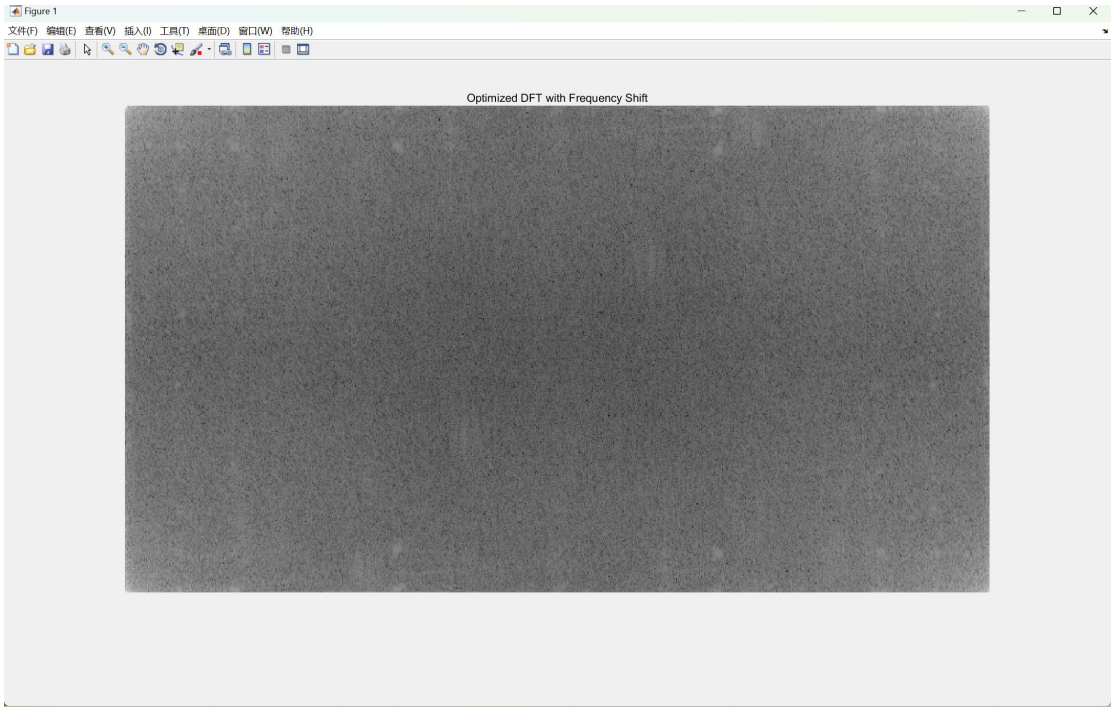
3.代码

% 读取图像

```
img = imread('D:\picture.bmp');

% 转换为灰度图
gray_img = rgb2gray(img);

% 优化后的二维傅里叶变换
dft_img = optimized_fft2(gray_img);

% 将频谱图的原点移至中心
shifted_dft = my_fftshift(dft_img);

% 显示频谱图
figure;
imshow(log(1 + abs(shifted_dft)), []);  % 使用对数尺度以增强可视化效果
title('Optimized DFT with Frequency Shift');

% --------- 优化后的二维傅里叶变换 (2D DFT) ----------
function F = optimized_fft2(img)
    [M, N] = size(img);  % 获取图像大小

    % 创建网格以便计算傅里叶变换的指数项
    [X, U] = meshgrid(0:M-1, 0:M-1);  % M x M 大小的网格
    [Y, V] = meshgrid(0:N-1, 0:N-1);  % N x N 大小的网格

    % 预先计算指数矩阵 (用于行和列的变换)
    exp_M = exp(-2j * pi * U .* X / M);
    exp_N = exp(-2j * pi * V .* Y / N);

    % 转换图像为 double 类型
    img = double(img);

    % 先沿行方向进行傅里叶变换，再沿列方向
    F_temp = exp_M * img;  % 沿行方向进行傅里叶变换
    F = F_temp * exp_N;      % 沿列方向进行傅里叶变换
end

% --------- 手动实现傅里叶变换频谱移位 (类似 fftshift) ----------
function shifted_F = my_fftshift(F)
    [M, N] = size(F);
    shifted_F = zeros(M, N);  % 初始化频谱移位矩阵

    % 频率原点移至图像中心
    for u = 1:M
        for v = 1:N
```

```
            % 将每个点乘以 (-1)^(u+v) 实现频率移位
            shifted_F(u, v) = F(u, v) * (-1)^(u + v);
        end
    end
end
```

4.实验结果



5.实验结论

通过手动实现傅里叶变换，掌握了频域分析的基本原理。频率移位将频谱中心化，有助于观察图像的频率特性，尤其是图像的低频和高频分量。