中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

学生姓名：　丁晓琪

学生学号：　22336057

专业名称：　计科

一：实现二维快速傅里叶变换及逆变换

1. 一维快速傅里叶变换：

- 理论：

$$一维傅里叶变换：F(u,v)=\sum_{x=0}^{M-1}f(x)e^{-j2\pi(\frac{ux}{M})}$$

$$令W_M^{ux}=(W_M)^{ux}=e^{-j2\pi ux/M}$$

$$令K=\frac{M}{2}$$

$$令F_{even}(u)=\sum_{x=0}^{K-1}f(2x)W_K^{ux}$$

$$令F_{odd}(u)=\sum_{x=0}^{K-1}f(2x+1)W_K^{ux}$$

$$u=0,1,\ldots K-1$$

$$则F(u)=F_{even}(u)+F_{odd}(u)W_{2K}^{u}$$

$$则F(u+K)=F_{even}(u)-F_{odd}(u)W_{2K}^{u}$$

- 实现：递归实现。
  - 一维FFT函数实现时：输入为f(x)的数组matrix[x],输出为包含所有u取值的F(u)的数组result[u]。因为快速傅里叶变换就是计算一次$F_{even}(u)$,$F_{odd}(u)$,能方便地得到$F(u),F(u+K)$，$F(u),F(u+K)$同时计算
  - 为了求出离散的f(x)（=matrix[x]）的傅里叶变换F(u)和F(u+K),需要计算$F_{even}(x)$和$F_{odd}(u)$。

```
#一维FFT：递归实现
def OFFT(matrix):
    M=len(matrix)
    if M==1:
        return matrix
    F_even=OFFT(matrix[0::2]) #获得偶数索引
    F_odd=OFFT(matrix[1::2]) #获得基数索引

    result=np.zeros(M,dtype=complex)
    for u in range(M//2):
        W_2k=np.exp(-2j*np.pi*u/M)
        result[u]=F_even[u]+W_2k*F_odd[u]
        result[u+M//2]=F_even[u]-W_2k*F_odd[u]
    return result
```

2. 二维快速傅里叶变换：

- 理论：将二维快速傅里叶变换转换为两次一维快速傅里叶变换,先对大括号内的做一次一维快速傅里叶变换，再对剩下的结果做一次傅里叶变换

$$//二维傅里叶变化：设图像x范围为0-M,y范围为0-N$$

$$F(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1}f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

$$令W_M^{ux}=(W_M)^{ux}=e^{-j2\pi ux/M}$$

$$则F(u,v)=\sum_{y=0}^{N-1}\{\sum_{x=0}^{M-1}f(x,y)\cdot W_M^{ux}\}\cdot W_N^{vy}$$

- 实现：

```
#二维FFT函数实现
```

```python
def DFFT(matrix):
    M_rows,M_cols=matrix.shape
    result = np.zeros((M_rows, M_cols), dtype=complex)
    #对x的每一行应用FFT，存在reslut中，result中行是u,列是y
    for i in range(M_rows):
        result[i,:]=OFFT(matrix[i,:])

    #对每一列执行FFT
    for j in range(M_cols):
        colum=result[:,j]
        result[:,j]=OFFT(colum)

    return result
```

3. 一维快速傅里叶逆变换：

- 理论：实际上就是一维快速傅里叶变换将e的指数的负号去除

$$\text{一维傅里叶逆变换：} f(x,y)=\sum_{u=0}^{M-1}F(u)e^{j2\pi(\frac{ux}{M})}$$

$$\text{令}W_M^{ux}=(W_M)^{ux}=e^{j2\pi ux/M}$$

$$\text{令}K=\frac{M}{2}$$

$$\text{令}f_{even}(x)=\sum_{u=0}^{K-1}F(2u)W_K^{ux}$$

$$\text{令}f_{odd}(x)=\sum_{u=0}^{K-1}F(2u+1)W_K^{ux}$$

$$x=0,1,\ldots K-1$$

$$\text{则}f(x)=f_{even}(x)+f_{odd}(x)W_{2K}^{u}$$

$$\text{则}f(x+K)=f_{even}(x)-f_{odd}(x)W_{2K}^{u}$$

- 实现：递归法和一维FFT基本一致，需要去除e的指数的负号

```python
#一维IFFT逆变换：递归实现
def OIFFT(matrix):
    M=len(matrix)
    if M==1:
        return matrix

    F_even=OIFFT(matrix[0::2]) #获得偶数索引
    F_odd=OIFFT(matrix[1::2])

    result=np.zeros(M,dtype=complex)
    for u in range(M//2):
        W_2k=np.exp(2j*np.pi*u/M)
        result[u]=F_even[u]+W_2k*F_odd[u]
        result[u+M//2]=F_even[u]-W_2k*F_odd[u]
    return result

```

4. 二维快速傅里叶逆变换

- 理论：和二维快速傅里叶变换基本一致，但是注意多了系数和将e的指数负号去除

$$//二维傅里叶变化：设u范围为0-M, v范围为0-N$$
$$f(x,y)=\frac{1}{MN}\sum_{v=0}^{N-1}\sum_{u=0}^{M-1}F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$
$$令W_M^{ux}=(W_M)^{ux}=e^{j2\pi ux/M}$$
$$则f(x,y)=\sum_{v=0}^{N-1}\{\sum_{u=0}^{M-1}F(u,v)\cdot W_M^{ux}\}\cdot W_N^{vy}$$

- 实现：先对行u做一维快速傅里叶逆变换，再对列v做一维快速傅里叶逆变换

```
#二维FFT逆变换函数实现
def DIFFT(matrix):
    M_rows,M_cols=matrix.shape
    result = np.zeros((M_rows, M_cols), dtype=complex)
    #对u的每一行应用IFFT，存在reslut中，result中行是x,列是v
    for i in range(M_rows):
        result[i,:]=OIFFT(matrix[i,:])

    #对每一列执行IFFT
    for j in range(M_cols):
        colum=result[:,j]
        result[:,j]=OIFFT(colum)

    return result/(M_cols*M_rows)
```

二：使用高斯低通滤波处理图像

1. 对图像进行零填充：

- 注意：图像初始大小为500*500，不符合要求（默认图像的长宽都是2的幂次方），此时需要先做边界零填充，在上下左右填充6行0，使得图像大小变为512*512

```
gray_matrix= np.pad(gray_matrix, ((6, 6), (6, 6)), mode='constant', constant_values=0)# 要转为512*512
```

为了避免图像卷积混叠，需要对图像填充0，使得大小从M*N变为2M*2N，且原图像在扩充后的图像的左上角

```
   padded_image_array = np.zeros((2*M, 2*N),    dtype=gray_matrix.dtype)
    padded_image_array[:M, :N] = gray_matrix
```

2. 频谱中心化：

要将频谱中心从(0,0)转移到($\frac{M}{2}$, $\frac{N}{2}$),也就是对空域乘$(-1)^{(x+y)}$

```
    indices_sum = np.indices((2*M, 2*N)).sum(axis=0)
    power_of_minus_one = (-1) ** indices_sum
    padded_image_array=power_of_minus_one*padded_image_array
```

3. 用二维快速傅里叶变换计算图像的频域分布F(u,v)

```
    ##傅里叶变化
    DFT_matrix=DFFT(padded_image_array)
```

4. 用高斯滤波器H(u,v)*F(u,v),得到滤波后频域分布

- 高斯低通滤波理论:

  $$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

- 高斯低通滤波实现:

```
#输入频域F（u，v）输出滤波后的F（u,v），截止频率D0
def GLPF(matrix,D0):
    M_rows,M_cols=matrix.shape

    ##构造滤波函数
    center_i, center_j = M_rows // 2, M_cols// 2
    H = np.zeros((M_rows, M_cols))
    for i in range(M_rows):
        for j in range(M_cols):
            # 计算当前点到频域中心的距离D
            D = np.sqrt((i - center_i)**2 + (j - center_j)**2)
            # 应用高斯函数计算响应值
            H[i, j] = np.exp(-(D**2) / (2 * D0**2))
    ##

    filtered_dft_matrix = matrix*H
    return filtered_dft_matrix
```

- 实现用截止频率为10的高斯低通对图像滤波

```
GLPF_DFT_matrix=GLPF(DFT_matrix,10)
```

5. 用二维快速傅里叶反变换对滤波后的频域分布转换为空域分布

减小计算机误差的影响对结果取实部

```
    # ##傅里叶反变换
    GLPF_matrix=DIFFT(GLPF_DFT_matrix)
    # ##取实部
    GLPF_matrix = np.real(GLPF_matrix)
```

6. 对得到的滤波后的图像的空域分布去中心化

将图像频域的中心点移动到(0,0)，也即对图像的空域分布乘$(-1)^{x+y}$

```
#去中心化
GLPF_matrix*=power_of_minus_one
```

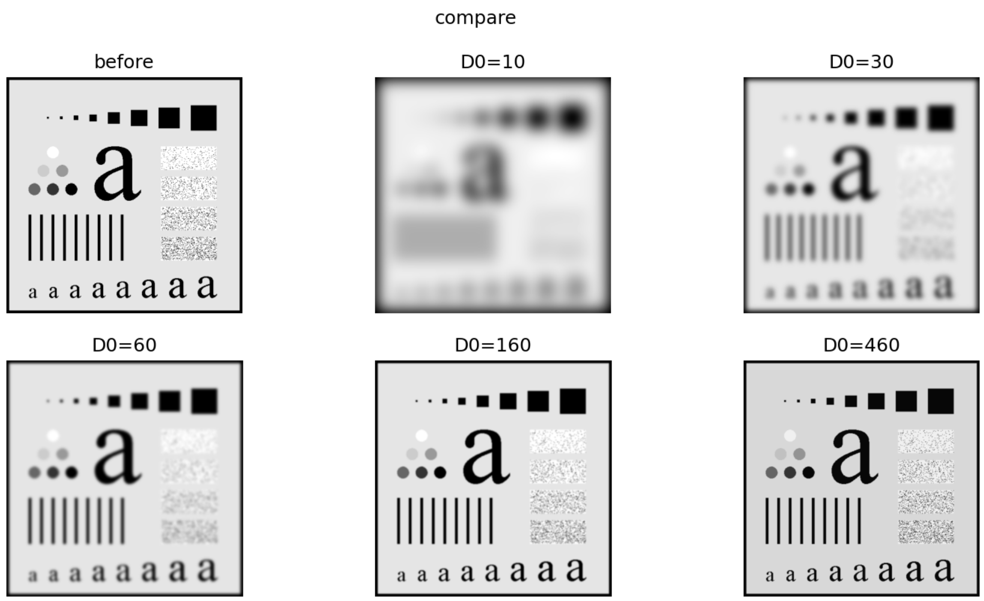
7. 提取出滤波后的图像的左上角

由于之前的零填充扩充了图像大小，所以需要把图像的大小复原，即取出左上角的M*N大小的部分

```
#7 提取出左上角
final_matrix1=GLPF_matrix[:M,:N]
```

三：实验结果展示：

高斯低通滤波的截止频率分别为10，30，60，160，460：可以看到截止频率越高，保留的能量越高，图像越清晰，保留细节越多



高斯低通滤波截止频率为80时：