桂林电子科技大学

**实验2 傅立叶变换**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **傅立叶变换** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计算机科学与技术 | | |
| 学 号 | 2000301320 | | | 姓名 | | 靳志凌 | | |
| 实验日期 | 2022 | 年 | 10 | | 月 | | 14 | 日 |
|  |  | | | | | | | |

## 一、实验目的

掌握傅立叶变换的基本原理和实现算法。

1. 能根据提供的图像数据，对图像进行变换，能解释得到变换结果的基本原理。

2. 能用傅立叶变换变换、逆变换还原图像。

## 二、实验内容与要求

1. 实现傅立叶变换及逆变换的算法。
2. 调入图像，对图像做傅里叶变换及频谱中心搬移，并显示。
3. 使用逆变换对上一步的数据进行处理，变显示。
4. 调出MATLAB自带傅里叶变换、逆变换函数，与你编写代码的效果进行比较。
5. 讨论不同的图像内容与频谱之间的对应关系。

## 三、实验环境

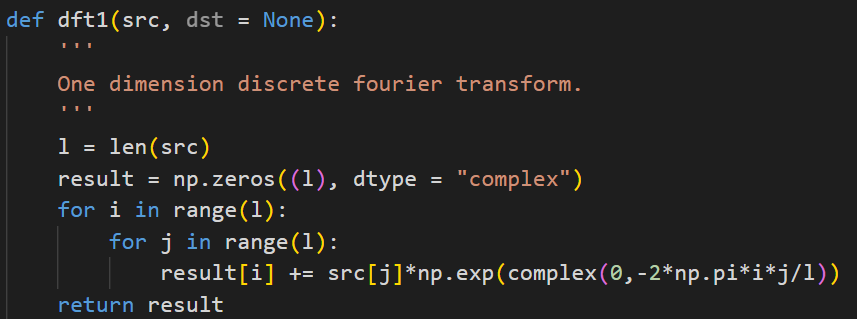
在实验室进行实验，学生可选Matlab、Python等等语言实现。

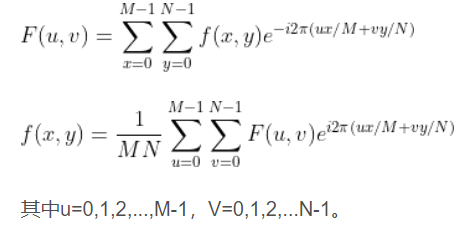
## 四、实验步骤

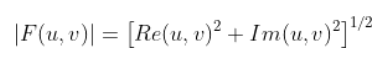
1.本次实验主要涉及到dft(src,dst=None),fft1(src,dst=None),iffft1(src),

Dec2Bin\_Inverse2Dec(n,m)以及Myfft2(f)等函数的编写。

f(x)=\frac{1}{M}\sum_{u=0}^{M-1}F(u)e^{i2\pi ux/M} , x = 0,1,2,...,M-1F(u)=\sum_{x=0}^{M-1}f(x)e^{-i2\pi ux/M},u=0,1,2,...M-1首先是dft(src,dst=None)函数的编写。由理论知识得出一维的离散函数的傅里叶变换和逆变换的公式如下：

根据计算公式可以写出dft1()函数，即如下代码：

在数字图像处理中我们关心的自然是，二维离散函数的傅里叶变换，直接给出二维离散傅里叶变换(Discrete Fourier Transform, DFT)的公式：

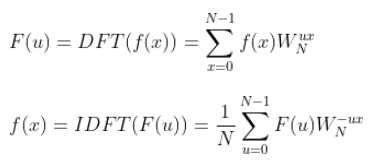
下面是傅里叶变换的幅度谱：

由幅度谱的定义可知，幅度谱具有对称性。幅度谱又叫频率谱，是图像增强钟关心的主要对象。频域下每一点的幅值都表示该频率的正弦平面波在叠加中所占的比例。相位谱则是隐含了实部Re和虚部Im之前的比例关系，因此和图像结构息息相关。但是在实际使用时，DFT的直接实现效率较低，所以有人提出了快速傅里叶变换(Fast Fourier Transform,FFT)，提高了计算的速度。

按时间抽取的FFT算法，是基于将输入序列f(x)分解(抽取)成较短序列，然后从这些序列的DFT中求得输入序列F(u)的方法。由于抽取后的较短序列仍然可分，所以最终仅需要计算一个很短序列的DFT。在这种算法中，我们首要关注的就是序列长度为2的整数次幂的序列，这种DFT运算称之为基-2DFF。

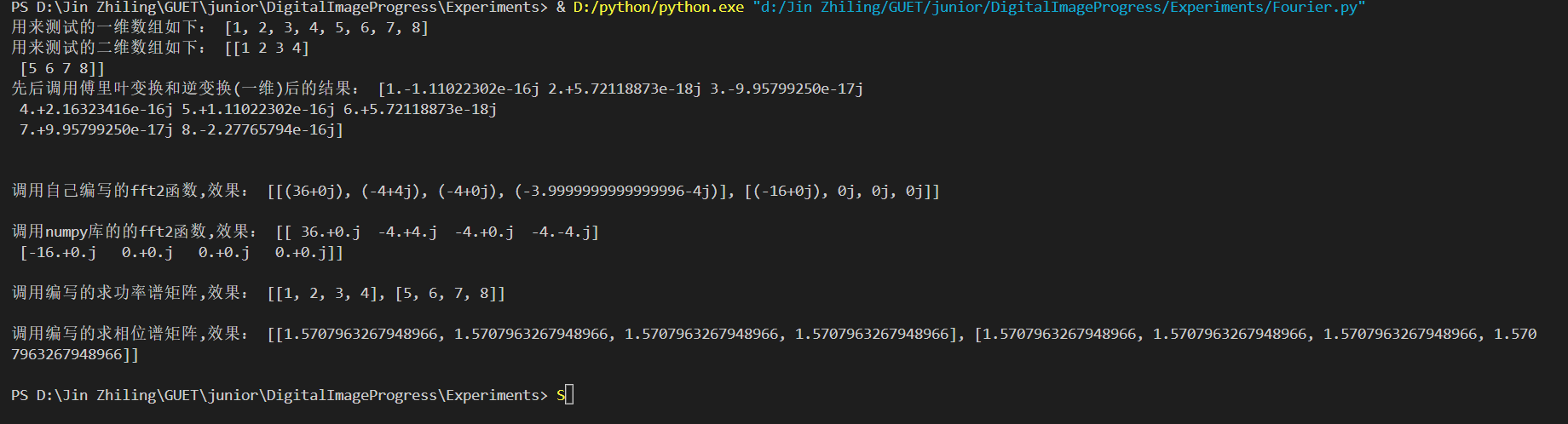
举个简单的例子，现在输入序列f(x)的长度是8。那么我们按序号来分成奇数项1，3，5，7和偶数0，2，4，6，然后计算各自的DFT。对于1，3，5，7这个四个数组成的序列的DFT，可以把3和7归入奇数项，1、5归入偶数项，再去分别计算DFT；对于0，2，4，6组成的序列的DFT，可以由0，4组成的奇数项的DTF，以及偶数项2，6组成序列的DFT来计算得到。然后取3，7为例，3是相当于偶数项，7是奇数项，已经无法在分割了，此时才开始根据定义来分别计算DFT。这就是基-2FFT的思路。

其中的函数 Dec2Bin\_Inverse2Dec(n,m)，功能是将01234567这样子排序的数组，根据输出时的新的顺序去调换成04261537。这是蝶形算法。

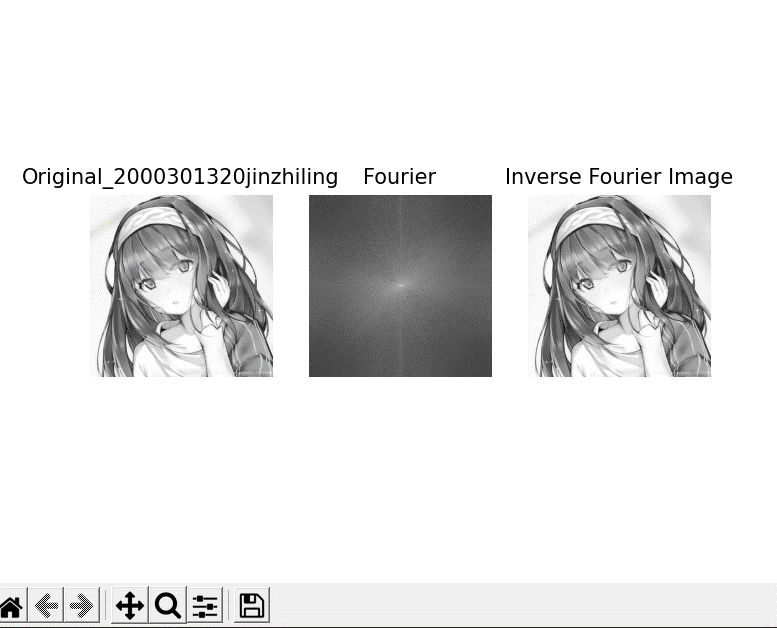
接着就是快速傅里叶逆变换了。首先观察两个公式：

可以发现，如果将DFT算子中的ux换成-ux，并在前乘以1/N，即可以得到IDFT的算子。这里一开始看不是很理解，因为变量一个u一个x混在一起，但后来仔细想，u、x也充其量是一个序号，因为我们是离散的序列。那么只要注意形式为主。可以推导离散反傅里叶变换的公式如下，只需先将F(u)取共轭就可以直接使用FFT算法计算IFFT。

至于二维的离散傅里叶变换，是在给定二维数组的每个维度上依次执行一维FFT，并且使用“原位运算”的方法，所以最终得到的结果也是一个二维数组。



由运行结果可知，自己编写的一维傅里叶变换和逆变换函数可以达到要求。编写的fft2函数与numpy库的fft2函数的运行结果误差较小，也可以达到要求。由于我采用了蝶形运算的算法，那个运算的数据需要满足是2的整数次幂个数，否则，在蝶形变换中会出现数组越界的问题。



代码：

import matplotlib.pyplot as plt

import numpy as np

import math

import copy

import cv2

def dft1(src, dst = None):

    '''

    One dimension discrete fourier transform.

    '''

    l = len(src)

    result = np.zeros((l), dtype = "complex")

    for i in range(l):

        for j in range(l):

            result[i] += src[j]\*np.exp(complex(0,-2\*np.pi\*i\*j/l))

    return result

def fft1(src, dst = None):

    l = len(src)

    n = int(math.log(l,2))

    bfsize = np.zeros((l), dtype = "complex")  #创建和src等大的一维复数数组

    for i in range(n + 1):

        if i == 0:

            for j in range(l):

                bfsize[j] = src[Dec2Bin\_Inverse2Dec(j, n)]

        else:

            tmp = copy.copy(bfsize)

            for j in range(l):

                pos = j%(pow(2,i))

                if pos < pow(2, i - 1):

                    bfsize[j] = tmp[j] + tmp[j + pow(2, i - 1)] \* np.exp(complex(0, -2\*np.pi\*pos/pow(2,i)))

                    bfsize[j + pow(2, i - 1)] = tmp[j] - tmp[j + pow(2, i - 1)] \* np.exp(complex(0, -2\*np.pi\*pos/(pow(2,i))))

    return bfsize

def ifft1(src):

    for i in range(len(src)):

        src[i] = complex(src[i].real, -src[i].imag)

    res = fft1(src)

    for i in range(len(res)):

        res[i] = complex(res[i].real, -res[i].imag)

    return res/len(res)

def Dec2Bin\_Inverse2Dec(n, m):

    '''

    Especially for fft.To find position.

    '''

    b = bin(n)[2:]

    if len(b) != m:

        b = "0"\*(m-len(b)) + b

    b = b[::-1]

    return int(b,2)

def Myfft2(src):

    h,w = src.shape

    hwft = [0 for i in range(h)]

    for row in range(h):

        hwft[row] = fft1(src[row])

    cft = [0 for i in range(w)]

    for col in range(w):

        FCol = [0 for i in range(h)]

        for row in range(h):

            FCol[row] = hwft[row][col]

        cft[col] = fft1(FCol)

    res = [ [0 for col in range(w)] for row in range(h)]

    for col in range(w):

        for row in range(h):

            res[row][col] = cft[col][row]

    return res

def fnGetPow(src):

    h = len(src)

    w = len(src[0])

    arrPow = [[0 for col in range(w)] for row in range(h)]

    arrPha = [[0 for col in range(w)] for row in range(h)]

    for row in range(h):

        for col in range(w):

            arrPow[row][col] = abs(src[row][col])

            v = src[row][col]

            vReal =  v.real

            vImag = v.imag

            arrPha[row][col] = math.atan2(vReal,vImag)

    return arrPow,arrPha

src = [1,2,3,4,5,6,7,8]

src2 = np.array([[1,2,3,4],[5,6,7,8]])

print('用来测试的一维数组如下：',src)

print('用来测试的二维数组如下：',src2)

print('先后调用傅里叶变换和逆变换(一维)后的结果：',ifft1(fft1(src)))

arrPower,arrPhase = fnGetPow(src2)

print('\n')

print('调用自己编写的fft2函数,效果：',Myfft2(src2),'\n')

print('调用numpy库的的fft2函数,效果：',np.fft.fft2(src2),'\n')

print('调用编写的求功率谱矩阵,效果：',arrPower,'\n')

print('调用编写的求相位谱矩阵,效果：',arrPhase,'\n')

#读取图像

img = cv2.imread('Images\girl.jpg', 0)

#傅里叶变换

dft = cv2.dft(np.float32(img), flags = cv2.DFT\_COMPLEX\_OUTPUT)

dftshift = np.fft.fftshift(dft)

res1= 20\*np.log(cv2.magnitude(dftshift[:,:,0], dftshift[:,:,1]))

#傅里叶逆变换

ishift = np.fft.ifftshift(dftshift)

iimg = cv2.idft(ishift)

res2 = cv2.magnitude(iimg[:,:,0], iimg[:,:,1])

#显示图像

plt.subplot(131), plt.imshow(img, 'gray'), plt.title('Original\_2000301320jinzhiling')

plt.axis('off')

plt.subplot(132), plt.imshow(res1, 'gray'), plt.title('Fourier')

plt.axis('off')

plt.subplot(133), plt.imshow(res2, 'gray'), plt.title('Inverse Fourier Image')

plt.axis('off')

plt.show()

## 五、问题记录和实验总结（必写）

实验中遇到的问题：在编写快速傅里叶变换函数的过程中，我采用了蝶形运算来简化计算。蝶形运算，2点 DFT 运算称为蝶形运算,而整个 FFT 就是由若干级 迭代 的蝶形运算组成,而且这种算法采用原位运算,故只需N个存储单元2. ∑∑ (2)式 (2)是FFT基4频域抽取算法的基本运算单元,一般称为蝶形运算. 第一列蝶形运算只有一种类型: 系数 ，参加运算的两个数据点间距为1。 第二列有两种类型的蝶形运算：系数分别为 ,参加蝶形运算的两个数据点的 间距 等于2。在用图片数组进行自己编写的fft2函数运算中出现了数组越界的问题。后来我尝试把图片的格式硬规定为512\*512规格，但还是出现数组越界的问题。

通过本次的实验，我了解到很多知识。尤其是新颖的思路，傅里叶变换将一件事物从时间域上的描述转移到频率域上的描述，在频率域上的描述可以将对于某些在时间域上的进行的极其困难的操作转化为十分简单的事情。关于这傅里叶变换我也观看了许多的相关视频，尽管现在对傅里叶变换仍感觉模糊，但我还是学习到了许多。