数图第三次作业

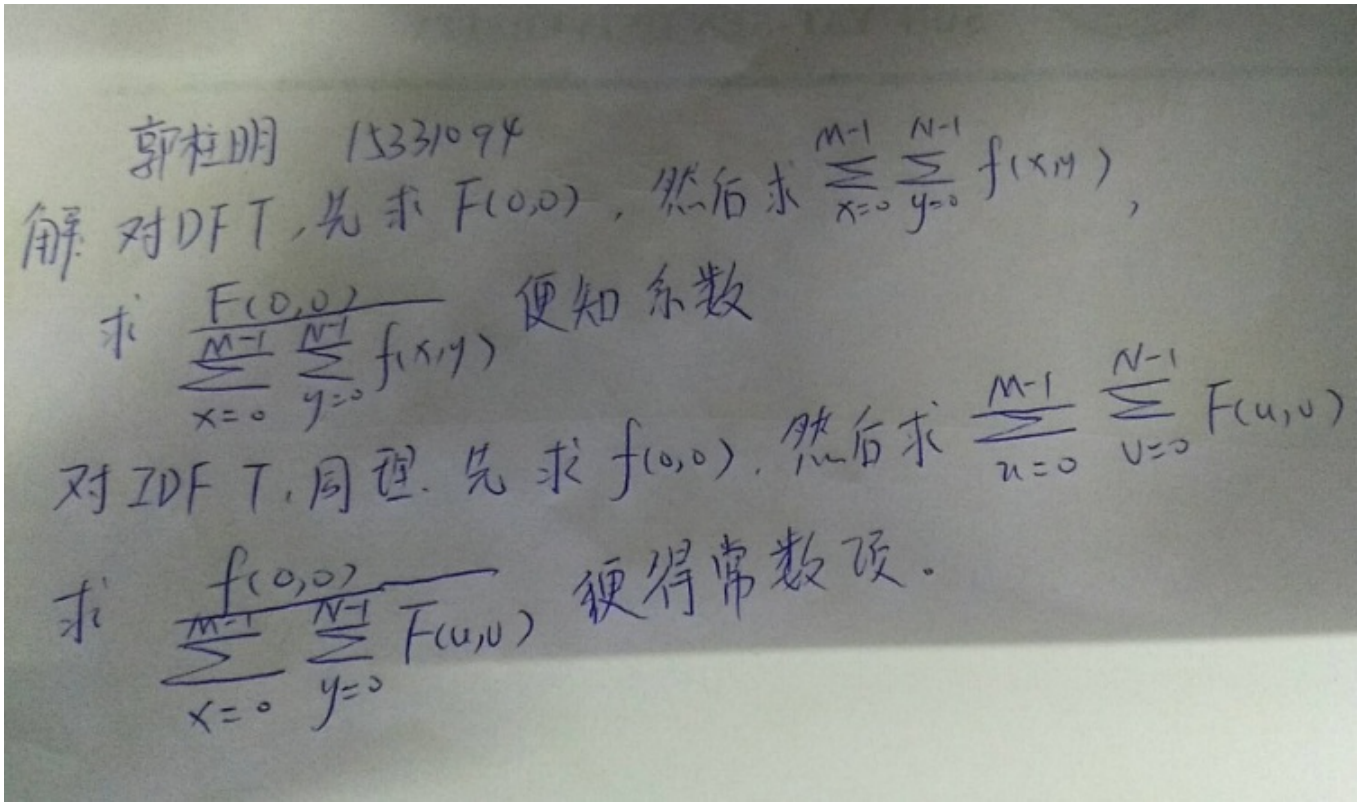
数图作业

郭柱明 15331094

1

1.1

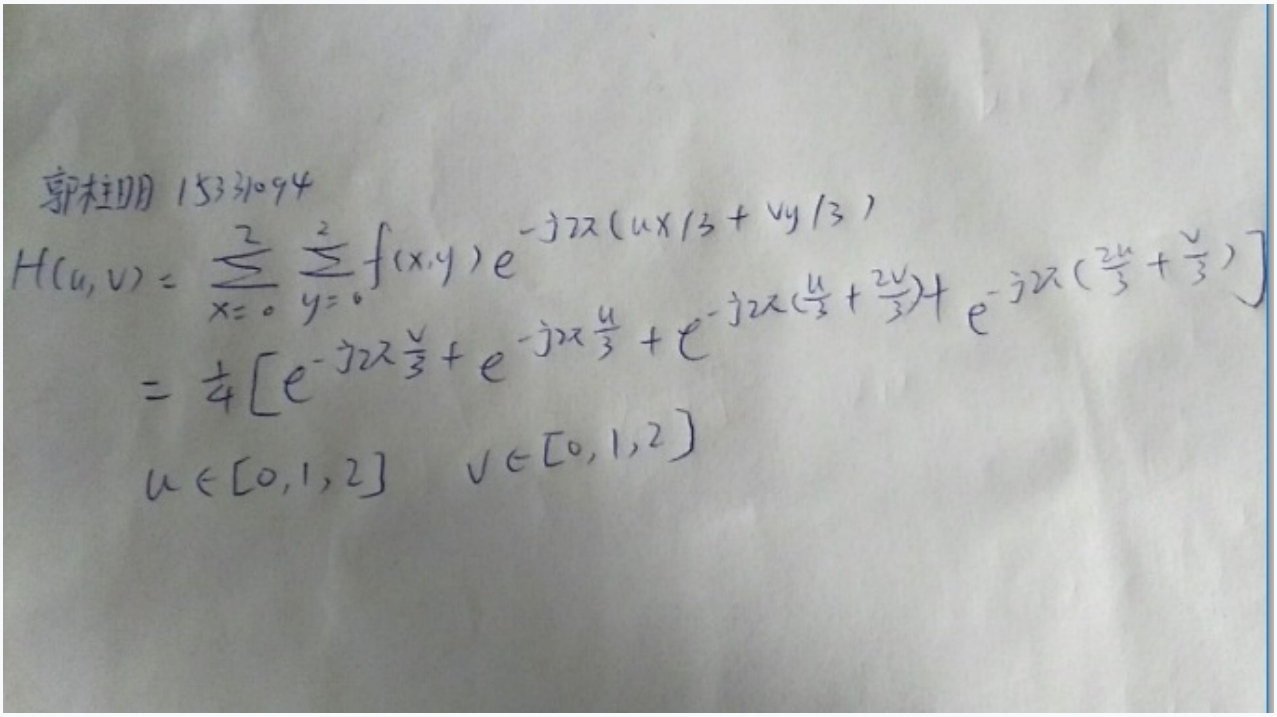
> 答：

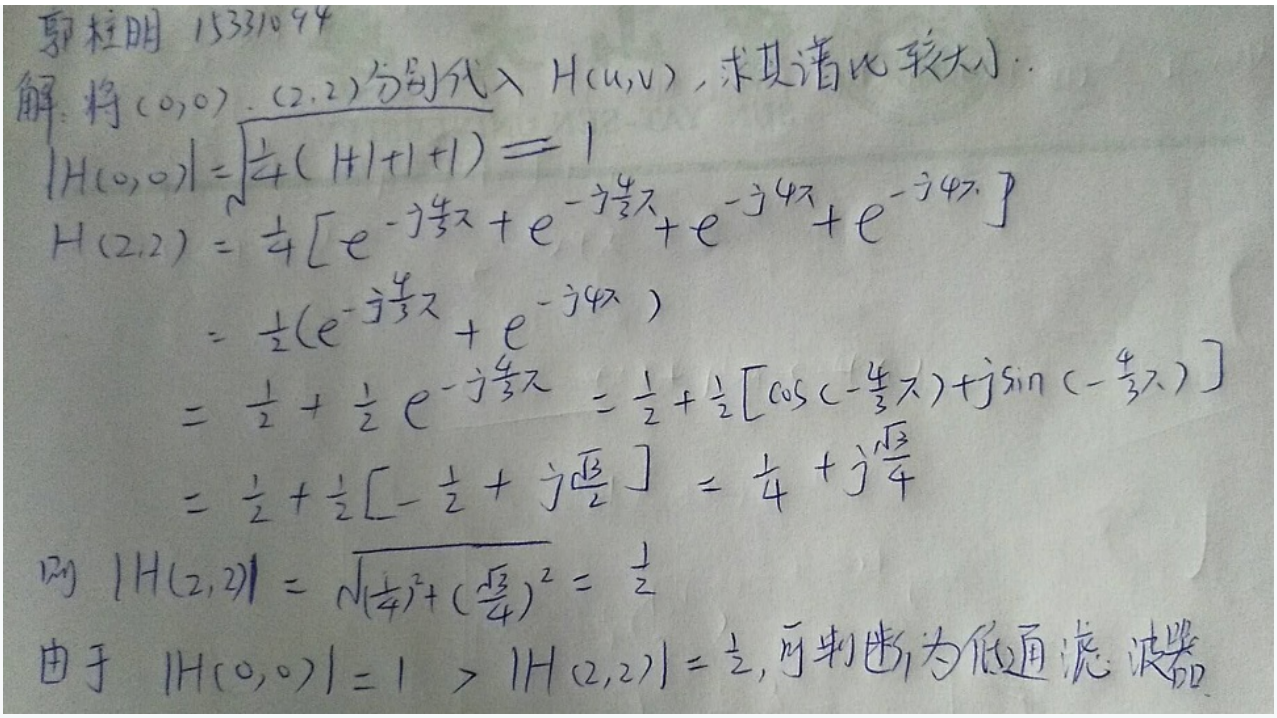


1.2

> 答：两幅图像的傅里叶频谱是相同的，因为c可以由b中的非0部分平移得到，由傅里叶频谱的平移性质可以知道，两者的傅里叶频谱相同。

1.3

> 答：
> 1.
> 
>
>
> 2.
>
> 低通滤波器，老师在课上说过，从图像上看，如果靠近原点的地方的频谱比较大，u，v越大，频谱越小则为低通滤波器



2

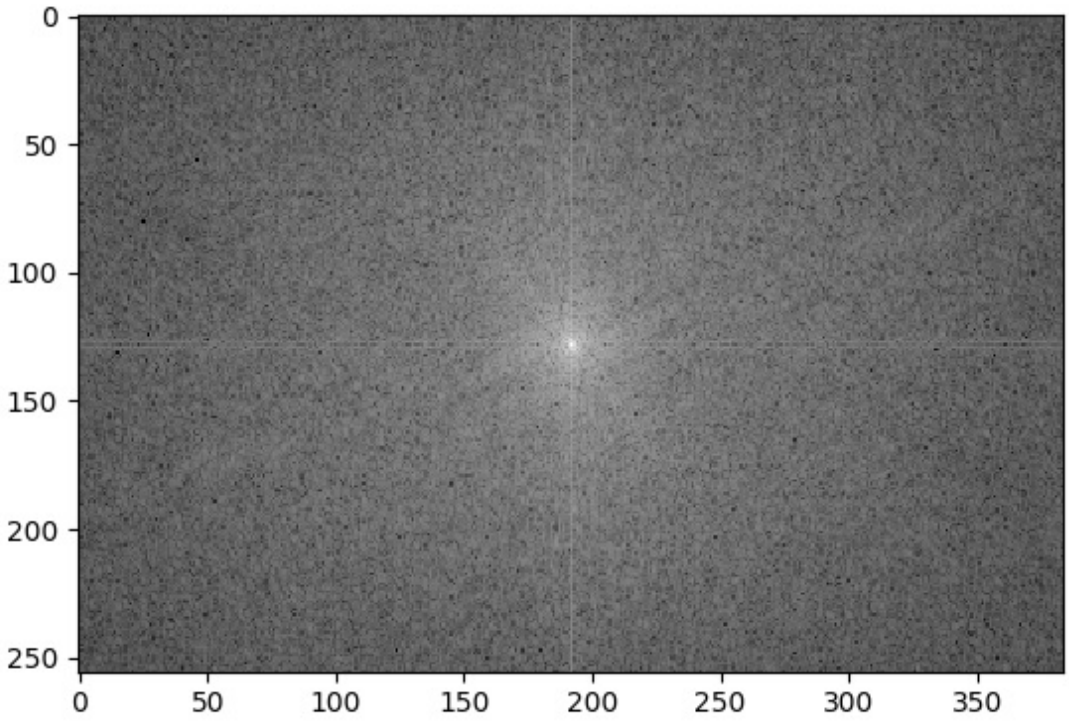
2.2

代码包含的python模块版本

模块	版本
numpy	1.13.1
PIL	4.1.1
matplotlib	2.0.2

1

> 答：



2

答：

3

答：因为学生我本来使用单纯的四层循环来求傅里叶变换发现耗时太多了，可行性很低，所以使用**离散傅里叶变换矩阵**

离散傅立叶变换矩阵是将离散傅立叶变换以矩阵乘法来表达的一种表示式。

定义 [编辑]

N点的离散傅立叶变换可以用一个$n \times m$的矩阵乘法来表示，即$X = Wx$，其中x是原始的输入信号，X是经过离散傅立叶变换得到的输出信号。 一个$n \times n$的变换矩阵W可以定义成$W = (\omega^{ij})_{i,j=0,\ldots,N-1}/\sqrt{N}$，或等效如下：

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix},$$

其中ω是1的n次方根的主值（primitive nth root of unity），大小为$e^{\frac{-2\pi i}{N}}$。需要注意的是在总和前面的正规化因数$\frac{1}{\sqrt{N}}$，还有ω中指数的正负号是依据惯例，并且会因为处理的方法有所不同。以下所有的讨论考虑到大多数的细节变动且不论是否为一般惯例均适用之。唯一重要的是，正变换和逆变换有相反的指数正负号标志，而其正规化因数乘积为$\frac{1}{N}$。然而，这里为了使得最后的离散傅立叶变换矩阵结果正规化所选择的因数，在许多情况下都是通用的。

获取离散傅里叶变换矩阵，根据参数flags。如果flags为dft，则返回**离散傅里叶变换矩阵**，如果为idft，则返回离散傅里叶变换矩阵的**逆矩阵**

```
#离散傅里叶变换矩阵或者傅里叶逆变换矩阵
def get_matrix(length, flags):
    i, j = numpy.meshgrid(numpy.arange(length),numpy.arange(length))
    pr = numpy.exp(-2j * numpy.pi / length)
    #形成傅里叶矩阵
    w = numpy.power(pr, i * j)
    #如果是傅里叶逆变换，则返回逆矩阵
    if flags == "idft":
        w = numpy.mat(w)
        w = numpy.array(w.I)
    return w
```

dft2d函数，如果参数image是一张图像的像素值矩阵，flags为dft，则计算其傅里叶变换的频谱图，如果image是一张频谱图的像素矩阵，flags为idft，则计算其傅里叶反变换后的图像

```
def dft2d(image, flags):
    heigh, width = image.shape
    output = numpy.zeros((heigh,width),numpy.complex)
    #如果是计算傅里叶变换
    if flags == "dft":
        image_shift = numpy.zeros((heigh,width), dtype = numpy.double)
        r = numpy.zeros((heigh,width),numpy.double)
        for x in range(heigh):
```

```
            for y in range(width):
                #图像每个像素点乘上-1的x+y次方，使得频谱图中心化
                image_shift[x, y] = image[x, y] * math.pow(-1, x + y )
        #矩阵相乘获得傅里叶变换
        output = get_matrix(heigh, "dft").dot(image_shift).dot(get_matrix(width, "dft"))
        for x in range(heigh):
            for y in range(width):
                #取模和取对数，以便显示
                r[x][y] = numpy.log(numpy.abs(output[x][y]))
        #绘制频谱图
        plt.imshow(r, "gray")
        plt.show()
        #返回频谱图的灰度矩阵
        return r
    #如果是傅里叶反变换
    elif flags == "idft":
        r = numpy.zeros((heigh, width))
        for x in range(heigh):
            for y in range(width):
                #对应傅里叶变换的取对数，即对每个像素值i求e^i
                output[x][y] = math.pow(numpy.e, image[x][y])
        #矩阵相乘求得傅里叶反变换
        output = get_matrix(heigh, "idft").dot(output).dot(get_matrix(width, "idft"))
        for x in range(heigh):
            for y in range(width):
                #获取实部，并且除以-1的x+y次方
                r[x][y] = output[x][y].real / math.pow(-1, x + y)
        #显示形成的图像
        Image.fromarray(r).show()
        Image.fromarray(r).convert("L").save("傅里叶反变换图像.png")
        #返回傅里叶反变换获得矩阵
        return r
```

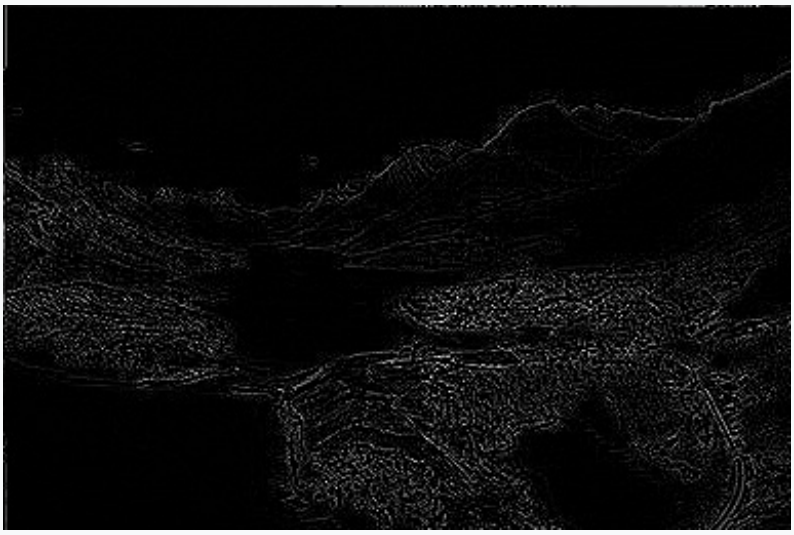
2.4

1

> 3 * 3

7 * 7

11 * 11



3

> 答：首先把滤波器的补0到大小跟图像的灰度矩阵大小一致

```
#滤波器空间域拓展为图像大小一致的矩阵
def resize_filter(input_img, my_filter):
    img_heigh, img_width = input_img.shape
    heigh, width = my_filter.shape
    result = numpy.zeros((img_heigh, img_width))
    for x in range(img_heigh):
        for y in range(img_width):
            if x <= heigh - 1 and y <= width - 1:
                result[x][y] = my_filter[x][y]
            else:
                result[x][y] = 0
    return result
```

分别求图像灰度矩阵的傅里叶变换结果的矩阵，以及补0之后的滤波器的傅里叶变换结果的矩阵，进行相乘，再进行傅里叶反变换，获得结果图像

```
def filter2d_freq(input_img, my_filter):
    #求图像灰度矩阵的傅里叶变换
    input_img = dft2d(input_img, "dft")
    #滤波器矩阵补0，扩展为跟图像灰度矩阵大小一致
    my_filter = resize_filter(input_img, my_filter)
    #求补0后的滤波器的傅里叶变换
    my_filter = dft2d(my_filter, "dft")
```

```
        print("my_filter", my_filter)
        #图像的傅里叶矩阵跟滤波器的傅里叶矩阵相乘
        result = input_img * my_filter
        #返回相乘结果的傅里叶反变换
        return dft2d(result, "idft")
```