1 Exercises

1.1 Rotation

1. Multiply Fig.1(a) f(x,y) by $(-1)^{x+y}$ to center its transform;

$$g(x,y) = f(x,y) * (-1)^{x+y} \Leftrightarrow F(u - \frac{M}{2}, v - \frac{N}{2})$$

2. Compute the DFT, F(u,v), of the image;

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

3. Taking the complex conjugate of the transform;

$$G^*(u,v) = [\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}]^* = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g^*(x,y)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

4. Computing the inverse discrete Fourier transform;

$$\Im^{-1}[G^*(u,v)] = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} G(u,v)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} G(u,v)e^{j2\pi(\frac{-ux}{M}+\frac{-vy}{N})} = g(-x,-y)$$

5. Multiplying the real part of the result by $(-1)^{x+y}$ where the real part is selected in order to ignore parasitic complex components resulting from computational inaccuracies.

We can see that these steps change the input image $f(x,y)$ to output image $f(-x,-y)$, which means that each point in the input image is mirrored about the origin(rotates 180°), thus producing the image on the right.

1.2 Fourier Sepctrum

把 0 添加到函数会在填充的边界和 0 之间形成不连续性，从而创建了一个不连续的函数，函数表现为在填充边界有着垂直和水平方向陡升/陡减的边，在傅里叶谱上就表现为沿垂直和水平轴的信号强度的显著增强。

1.3 Lowpass and Highpass

1. At any point (x,y) in the image, the response, g(x,y), of the filter is the sum of products of the filter coefficients and the image pixels encompassed by the filter:

$$\mathrm{g(x,y) = f(x,y+1) + f(x+1,y) + f(x-1,y) + f(x,y-1) + 2f(x,y)}$$

Then, using the following property:

$$\mathrm{f(x-x_0, y-y_0)} \Leftrightarrow F(u,v)e^{-j2\pi(\frac{ux_0}{M}+\frac{vy_0}{N})}$$

$$\mathrm{G(u,v) = \left[e^{\frac{j2\pi v}{N}} + e^{\frac{j2\pi u}{M}} + e^{-\frac{j2\pi u}{M}} + e^{-\frac{j2\pi v}{N}} + 2\right] F(u,v) = H(u,v)F(u,v)}$$

So the equivalent filter

$$\mathrm{H(u,v) = e^{\frac{j2\pi v}{N}} + e^{\frac{j2\pi u}{M}} + e^{-\frac{j2\pi u}{M}} + e^{-\frac{j2\pi v}{N}} + 2 = 2\cos\left(\frac{2\pi u}{M}\right) + 2\cos\left(\frac{2\pi v}{N}\right) + 2}$$

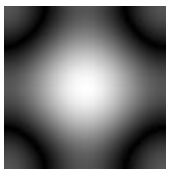
2. H(u,v) is composed of cosine function, we know cos(x) is a periodic function taking 2π as the period. When x=0, its maximun value is 1 and the values of cos(x) decrease as x move away from the origin when $|u| \leq \pi/2$. After that, cos(x) starts to rise to 1.

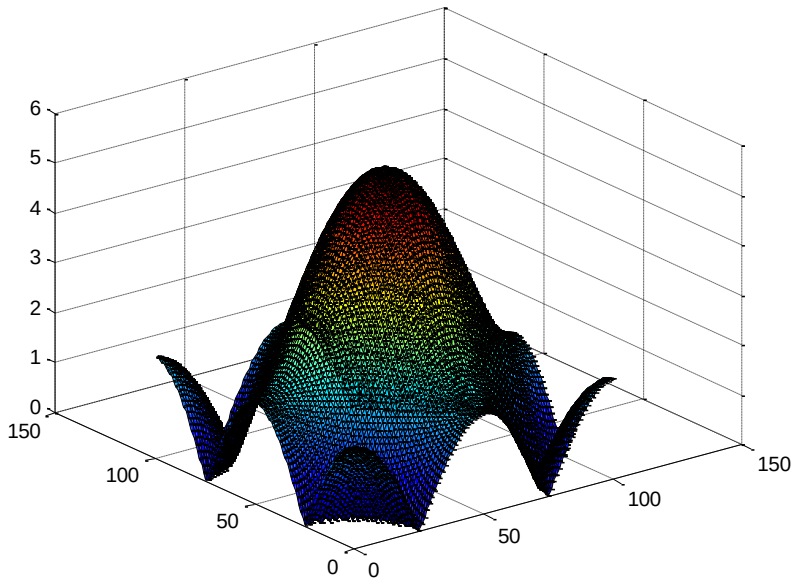
So H(u,v) acts like a low pass filter because at (u,v)=(0,0), H(u,v)=6.

$$H(u,v) = 2\cos\left(\frac{2\pi u}{M}\right) + 2\cos\left(\frac{2\pi v}{N}\right) + 2$$

The values of H(u,v) decrease as (u,v) move away from the origin when $|u| \leq M/4$ and $|v| \leq N/4$. However, it will rise after that. So it does not attenuate those high frequencies like a low pass filter should.

We can plot the magnitude of H using Matlab:





From the above image, we see that at very high frequencies(where both u and v are high), the magnitude of H starts to rise again.

1.4 Padding

From Section 4.7, we know that padding the two ends of the function is the same as padding one end, as long as the total number of zeros used is the same. This is because the process of zero padding is used to solve the problem of wraparound error in convolution.

Consider two functions, f(x) and h(x) composed of A and B samples, respectively. It can

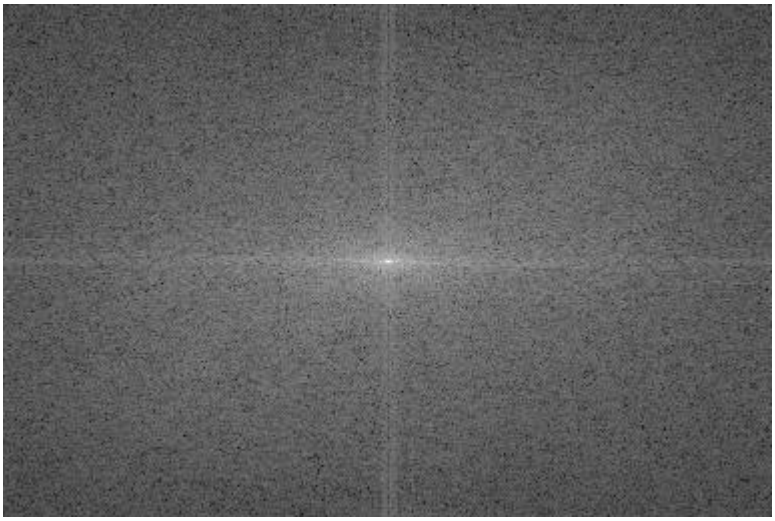
be shown that if we append zeros to both functions so that they have the same length, demoted by P, then wraparound is avoided by choosing $P \geq A + B - 1$.
The zeros could be appended also to the beginning of the functions, or they could be divided between the beginning and end of the functions.
In 2-D the situation is similar.

2 Programming Tasks

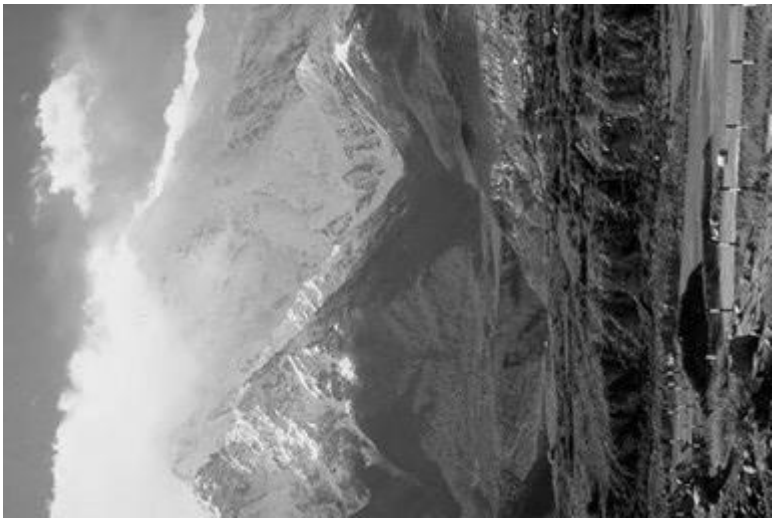
2.1 Pre-requirement

2.2 Fourier Transform

1. DFT



2. IDFT



3. Implementation
   相关的文件有以下五个：
   dft.m
   输入：大小为 M×N 的数字图像$f(x, y)$
   输出：二维傅里叶变换$F(u, v)$
   idft.m

输入：二维傅里叶变换$F(u,v)$

输出：傅里叶反变换$f(x,y)$

dft2d.m

输入 1：大小为 M×N 的数字图像 input_img

输入 2：flags 为-1 表示进行 DFT，否则进行 IDFT

输出：flags 为-1 时 output_img 为经过中心化和 DFT 变换的输出，

否则 output_img 为经过 IDFT 变换和取实数部分并去中心化的图像。

display_spectrum.m

输入：经过傅里叶变换的$F(u,v)$

输出：用于显示的傅里叶频谱图 spectrum

main_hw3.m

dft 部分：调用 dft2d(input_img,-1)获得经 DFT 变换的$F(u,v)$，调用 display_spectrum 获得显示的频谱图并显示出来。

idft 部分：调用 dft2d(input_img,1)获得经 IDFT 变换的图像并显示出来。

算法实现：

dft.m

傅里叶变换基于课本 4.11.1 的算法实现

根据 2-D DFT 的可分性来简化对 2-D DFT 的计算，

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

$$=\sum_{x=0}^{M-1}F(x,v)e^{-j2\pi\frac{ux}{M}}$$

其中，

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

$F(x,v)$遍历图上每一个像素，计算它所在的行的所有像素的灰度值和某虚数乘积之和。$F(u,v)$遍历图上每一个像素，计算它所在的列的所有像素的$F(x,v)$和某虚数乘积的和。也就是说，$f(x,y)$的二维 DFT 可以通过计算$f(x,y)$的每一行的一维变换，然后，沿着计算结果的每一列计算一维变换来得到。利用两个三重循环即可完成上述计算。

idft.m

利用上述的 DFT 算法来计算 IDFT，原理如下：

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

对上式的等号两边取复共轭，并且乘上MN，得到

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

上式等号右边的部分是$F^*(u,v)$的 DFT。因此，我们可以将$F^*(u,v)$代入 DFT 算法得到$MNf^*(x,y)$。对这个结果取复共轭并除MN就得到了$f(x,y)$，即$F(u,v)$的 IDTF。

dft2d.m

若进行 DFT 变换，首先将输入图像转化为 double 类型，再通过遍历每个像素点乘以$(-1)^{x+y}$来进行中心化以更好地显示图像，最后调用 dft 函数。

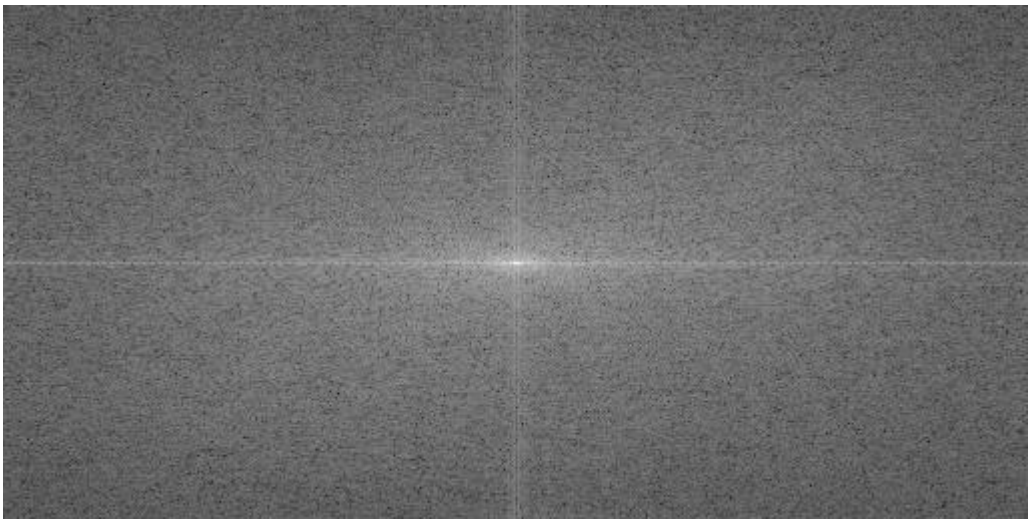
若进行 IDFT 变换，首先调用 idft 函数，再取实数部分，然后要通过遍历每个像素点乘以$(-1)^{x+y}$来去中心化以消除前一步做了中心化的影响，最后要将前面的结果进行 uint8 转换作为输出的图像（才能直接显示）。

display_spectrum.m

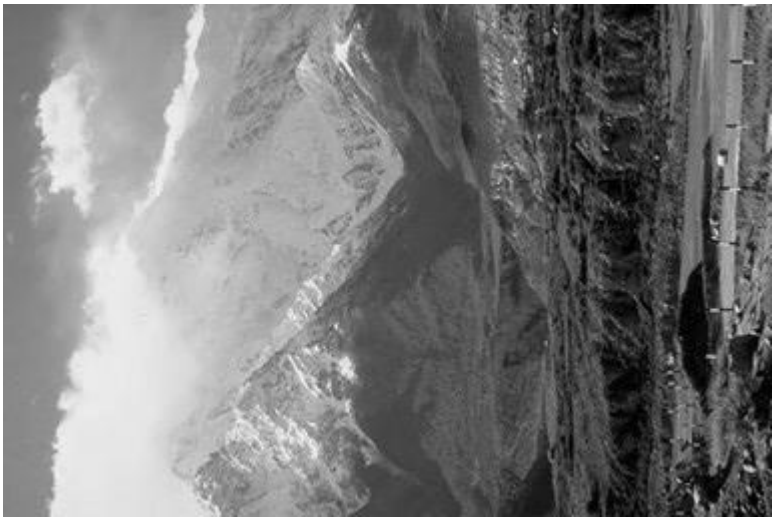
显示傅里叶变换图像分为三步：首先用 abs 方法实数化$F(u,v)$，第二计算频谱图的 log 变换，第三进行 Scaling 变换。

2.3 Bonus: Fast Fourier Tranform

1. FFT



2. IFFT



3. Why FFT have a lower time complexity than DFT?

   根据课本 4.11.3 节对快速傅里叶变换（FFT）的讨论，（以下符号及设定和课本一致）如果要直接实现 DFT，公式的大规模执行要求约$(\mathbf{MN})^{\mathbf{2}}$的求和及加法。快速傅里叶变换可将乘法和加法的次数降低到$\boldsymbol{MNlog_2MN}$。

   数学证明如下

   由于二维 DFT 可以使用逐次通过一维变换的方法来执行，因此我们仅需关注一个变量的 FFT。在一维上，我们看到一个 M 点变换可以通过把原始表达式分解为两部分来计算。采用**逐次加倍**的方法，可以推导出对任意正整数 n 计算这些参量完成 FFT 所需的乘法和加法次数的递归表达式为

   $$\mathrm{m(n)} = 2\mathrm{m(n-1)} + 2^{\mathrm{n-1}},\ \ \mathrm{n} \geq 1$$

   和

   $$\mathrm{a(n)} = 2\mathrm{a(n-1)} + 2^{\mathrm{n}},\ \ \mathrm{n} \geq 1$$

   其中，m(0)=0 且 a(0)=0，因为单点变换无须任何加法和乘法运算。

   对于任意的 2 的整数次幂的 M，一个两点变换来自两个单点变换，一个 4 点变换来自两个两点变换，以此类推，直到 M 为 2 的整数次幂的情形。可以证明

   $$\mathrm{m(n)} = \frac{1}{2}Mlog_2M$$

   和

   $$\mathrm{a(n)} = \mathrm{Mlog_2M}$$

   与直接计算一维 DFT 相比，FFT 的计算优势定义为

   $$\mathrm{c(M)} = \frac{\mathrm{M^2}}{Mlog_2M} = \frac{M}{\log_2 M}$$

   因为假定有$\mathrm{M} = 2^{\mathrm{n}}$，所以可以将上式写为关于 n 的形式：

   $$\mathrm{c(n)} = \frac{2^n}{n}$$

   从上面的推算可以看出，FFT 的时间复杂度远小于 DFT。

4. Implementation.

   文件参数定义和整体操作（如中心化、显示频谱图等）和 2.2 题类似，不再赘述，下面说明算法部分：

   基于课本 4.11.3 一维快速傅里叶变换算法实现的二维快速傅里叶变换：

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

$$=\sum_{x=0}^{M-1}F(x,v)e^{-j2\pi\frac{ux}{M}}=\sum_{x=0}^{M-1}F(x,v)W_M^{ux}$$

其中，

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

式中，u=0,1,…,M-1，v=0,1,…,N-1 其中，

$$\mathrm{W_M}=\mathrm{e^{-j2\pi/M}}$$

$$\mathrm{W_N}=\mathrm{e^{-j2\pi/N}}$$

且假设 M,N 具有如下形式：

$$\mathrm{M}=2^{n1}$$

$$\mathrm{N}=2^{n2}$$

其中 n1,n2 是正整数。因此，M,N 可表示为

$$\mathrm{M}=2\mathrm{K_1}$$

$$\mathrm{N}=2\mathrm{K_2}$$

$\mathrm{K_1},\mathrm{K_2}$也是正整数。代入得

$$F(u,v)=\sum_{x=0}^{2K-1}F(x,v)W_M^{ux}=\sum_{x=0}^{K-1}F(2x,v)W_{2K}^{u(2x)}+\sum_{x=0}^{K-1}F(2x+1,v)W_{2K}^{u(2x+1)}$$

然而，可以证明$W_{2K}^{u(2x)}=W_K^{ux}$，因此上式可以表示为

$$F(u,v)=\sum_{x=0}^{K-1}F(2x,v)W_K^{ux}+\sum_{x=0}^{K-1}F(2x+1,v)W_K^{ux}W_{2K}^{u}$$

定义

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

式中，u=0,1,2,…,K-1,和

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

式中，u=0,1,2,…,K-1,式子化简为

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

此外，因为$W_M^{u+M}=W_M^u$和$W_{2M}^{u+M}=-W_{2M}^u$，可以给出

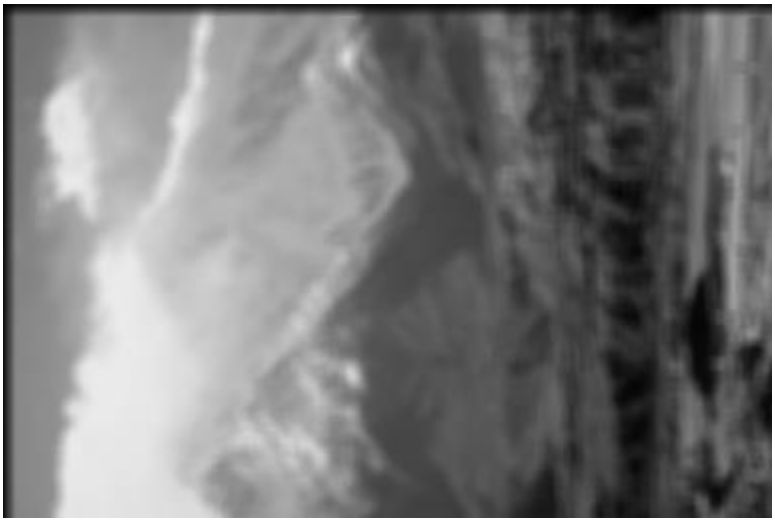
$$F(u+K,v)=F_{even}(u,v)-F_{odd}(u,v)W_{2K}^{u}$$

首先，如二维 DFT 的可分性一样，二维 FFT 也有相似的性质。算法的重点在于逐次加倍的思想。举例来说，一维中一个 M 点变换可以通过把原始表达式分解为两部分来计算。计算 F(u)的前一半要求计算公式中给出的两个(M/2)点变换。然后，将结果即$F_{even}(u)$和$F_{odd}(u)$的值代入得到$F(u),\mathrm{u}=0,1,2,\dots,(\mathrm{M}/2-1)$。

然后，可以直接计算$F(u)$的另一半，而不需要额外的变换计算。

2.4 Filtering in the Frequency Fomain

1. With 7*7 averaging filter.



2. With 3*3 Laplacian filter.

3. Implementation.

   依据课本 4.7.3 频率域滤波步骤：

   1）给定一幅大小为 M×N 的输入图像$f(x,y)$，得到填充参数 P 和 Q。典型地，我们选择 P=2M 和 Q=2N。

   2）对$f(x,y)$添加必要数量的 0，形成大小为 P×Q 的填充后的图像$f_p(x,y)$。

   3）用$(-1)^{x+y}$乘以$f_p(x,y)$移到其变换的中心。

   4）计算来自步骤 3 的图像的 DFT，得到$F(u,v)$。

   5）生成一个实的、对称的滤波函数$H(u,v)$，其大小为 P×Q，中心在$(P/2,Q/2)$处。如果$H(u,v)$由一个给定的空间滤波器$h(x,y)$生成，则可按如下方式形成$h_p(x,y)$：将该空间滤波器的大小填充为 P×Q，使用$(-1)^{x+y}$乘以扩展后的阵列，并计算结果的 DFT 来得到$H(u,v)$。

   用阵列相乘形成乘积$G(u,v)=H(u,v)F(u,v)$；即$G(i,k)=H(i,k)F(i,k)$.

   6）得到处理后的图像：

$$g_p(x,y) = \{real[\Im^{-1}[G(u,v)]]\}(-1)^{x+y}$$

其中，为忽略由于计算不准确导致的寄生复变量，选择了实部，下标 p 指出我们处理的是填充后的阵列。

7）通过从$g_p(x,y)$的左上象限提取 M×N 区域，得到最终处理结果$g(x,y)$。

我的实现和上述步骤一模一样（包括变量名称），其中 DFT、IDFT 的计算调用了 2.2 题实现的函数。

值得注意的是，步骤五中我们使用的第一是 7*7 均值滤波器，和上述步骤一致，第二是 3*3 拉普拉斯滤波器，最终结果为原图像减去计算的结果。这是因为对于均值滤波器，计算得到的加权值即输出图像的灰度值：

$$\text{output\_img}(r,c) = \text{intensity}(r,c)$$

对于拉普拉斯滤波器，输出图像的灰度值等于输入图像的灰度值加上 c 倍的加权值：

$$\text{output\_img}(r,c) = \text{input\_img}(r,c) + c \times \text{intensity}(r,c)$$

对于题目中应用的拉普拉斯滤波器，取 c=-1。