Homework 4

13331314 叶嘉祺

1.1 Rotation (10 Points)
Fig. 1(b) was generated by:
1. Multiplying Fig. 1(a) by (−1)^(x+y) ;
2. Computing the discrete Fourier transform;
3. Taking the complex conjugate of the transform;
4. Computing the inverse discrete Fourier transform;
5. Multiplying the real part of the result by (−1) x+y .
Explain (mathematically) why Fig. 1(b) appears as it does

Assume that the origin image is:

$$F = f(x, y)$$

1.Multiplying Fig. 1(a) by (−1) x+y :

$$F' = f'(x, y) = f(x, y) \cdot (-1)^{x+y}$$

2. Computing the discrete Fourier transform;

$$F^*_{Ft} = Ft(F') = Ft(f(x, y)(-1)^{x+y}) = f_{Ft}(j\omega_x, j\omega_y)$$

3. Taking the complex conjugate of the transform;

$$F'_{Ft} = F^*_{Ft} = f^*_{Ft}(j\omega_x, j\omega_y)$$

4. Computing the inverse discrete Fourier transform;

$$\because h'^*(x, y) \Leftarrow FT \Rightarrow h^*_{FT}(j\omega_x, j\omega_y)$$

$$\therefore g'(x, y) = invFT(f^*_{Ft}(j\omega_x, j\omega_y)) = f'^*(x, y) = f'(-x, -y)$$

5. Multiplying the real part of the result by (−1) x+y .

$$g(x, y) = g'(x, y) \times (-1)^{x+y} = f'(-x, -y) \times (-1)^{x+y}$$

So the result image is shown below:

$$g(x,y) = f(-x,-y)$$

1.2 Fourier Spectrum (10 Points)

The (centered) Fourier spectrum in Fig. 2(b) corresponds to the original image Fig. 2(a), and the (centered) Fourier spectrum in Fig. 2(d) was obtained after Fig. 2(a) was padded with zeros (shown in Fig. 2(c)). Explain the significant increase in signal strength along the vertical and horizontal axes of Fig. 2(d) compared with Fig. 2(b).

Fourier Spectrum has the following properties：

1. Considering the symmetry of the Fourier transform, in order to facilitate the display, the Fourier spectrum is usually based on the center of the image(the original point). Each part of the image is about the origin of symmetry.
2. The original point of the the Fourier spectrum is the “zero frequency point” which is the average gray value of the image containing the general information. And from center to the edge of the spectrum, the frequency is increasing and indicates the details of the image.
3. In the center of the frequency domain, the obvious change direction is vertical to the surface of the original image. In other words, if the obvious change direction in the original image is vertical, then the obvious change direction in frequency domain is horizontal.

For the image Figure2(a) and Figure2(c):

We padded zeros in Figure2(c), and also note that we add high frequency component for the edge in the image. Because we can see it clearly that the edge of the image is trend to 255 but the padding is 0. So from the edge to the padding, we got high frequency.

According to the properties I describe before, we can know that both the vertical and horizontal axes will be strengthened in frequency domain.

For both vertical and horizontal axes in original image, the edge makes a big change in the border of the padding in Figure2(c), but not in Figure3(a). As a result, we got a different Fourier Spectrum from Figure2(c) with more high frequency component in both vertical and horizontal axes.

1.3 Lowpass and Highpass (5 ∗ 2 = 10 Points)

1. Find the equivalent filter H(u, v) in the frequency domain for the following spatial filter:

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

For this filter:

$$g(x,y) = f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1) - f(x+1,y-1) - 2f(x+1,y) - f(x+1,y+1)$$

$$G(x,y) = (e^{\frac{-2j\pi u}{M}+\frac{-2j\pi v}{N}} + 2e^{\frac{-2j\pi u}{M}} + e^{\frac{-2j\pi u}{M}+\frac{2j\pi v}{N}} - e^{\frac{2j\pi u}{M}+\frac{-2j\pi v}{N}} - 2e^{\frac{2j\pi u}{M}} - e^{\frac{2j\pi u}{M}+\frac{2j\pi v}{N}}) \cdot F(u,v) = H(u,v)\cdot F(u,v)$$

Also in other way, Assume the matrix is f(x,y)

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

```
filter = [[1,2,1],[0,0,0],[-1,-2,-1]]
filter =  np.abs(np.fft.fftshift(np.fft.fft2(a)))
```

Result not centerlize:

$$\begin{matrix} 0 & 0 & 0 \\ 6.92820323 & 1.73205081 & 1.73205081 \\ 6.92820323 & 1.73205081 & 1.73205081 \end{matrix}$$

Result after centerlize H(u,v):

$$\begin{matrix} 1.73205081 & 6.92820323 & 1.73205081 \\ 0 & 0 & 0 \\ 1.73205081 & 6.92820323 & 1.73205081 \end{matrix}$$

2. Is H(u, v) a low-pass filter or a high-pass filter? Prove it mathematically.
It's a high-pass filter.

Suppose that we have a high frequency image g(x,y) and it's Fourier transform(after centerlize) is G(u,v) (3*3)

Since g(x,y) is a high frequency image, G(u,v)'s edge which indicates the high frequency components is not zero:
Assume G(u,v):

$$\begin{matrix} HIGH & HIGH & HIGH \\ HIGH & LOW & HIGH \\ HIGH & HIGH & HIGH \end{matrix}$$

We get I(u,v):

$$I(u,v) = G(u,v) \cdot H(u,v)$$

we get the flowing image represented in frequency domain

$$\begin{matrix} HIGH & 6HIGH & HIGH \\ 0 & 0 & 0 \\ HIGH & 6HIGH & HIGH \end{matrix}$$

Now we can see that, the high frequency component in the horizon axes is reserved. As a result, it's a high pass filter.

1.4 Exchangeability (10 Points)
Suppose the high-frequency-emphasis filtering is applied to an image for edge sharpening, and then the histogram equalization is applied to enhance contrast. Theoretically, would the result be the same if the order of these two operations were reversed? Prove it mathematically. Hint: Refer to Chapter 2.6.2 "Linear versus Nonlinear Operations".

No, not the same.
Linear operations satisfied the following formula:

$$H(a_i f_i(x,y) + a_j f_j(x,y)) = a_i H(a_i f_i(x,y)) + a_j H(a_j f_j(x,y))$$

We can prove that high-frequency-emphasis filtering is a linear operation.

$$H(aF(u,v) + bG(u,v)) = (aF(u,v) + bG(u,v)) \cdot H(u,v) = aF(u,v) \cdot H(u,v) + bG(u,v) \cdot H(u,v)$$

Note that Fourier transform is also linear.
So high-frequency-emphasis filtering is linear.

For histogram equalization:

$$s = T(r) = \int_0^r P_r(\omega) d\omega$$

Obvious，

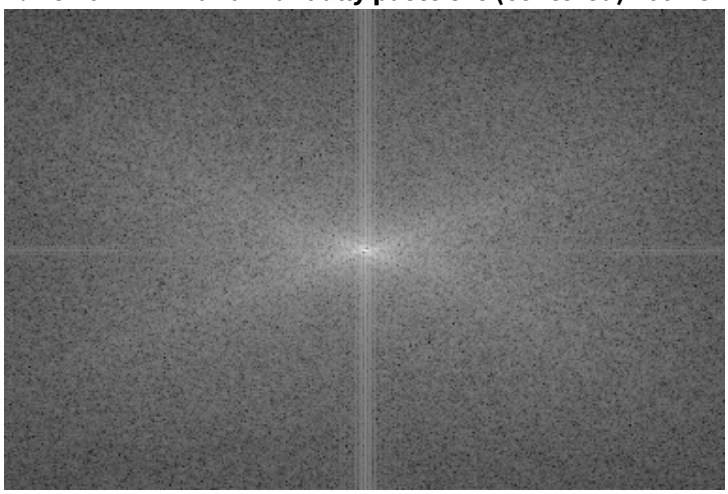
$$T(af(x) + bg(x)) \neq aT(f(x)) + bT(g(x))$$

So histogram equalization is no linear transform.

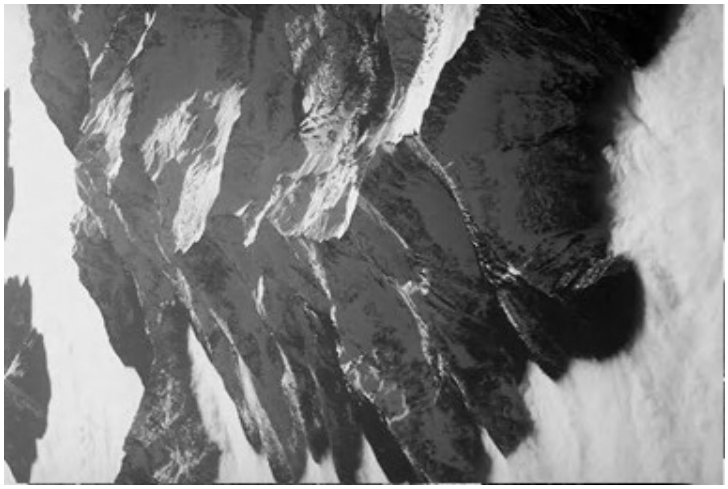
As a result, the result **wil not** be the same if the order of these two operations were reversed for which one operation is linear and the other is not.

2 Programming Tasks

2.2 Fourier Transform (30 Points)
Write a function to perform 2-D Discrete Fourier Transform (DFT) or Inverse Discrete Fourier Transform (IDFT). The function prototype is “dft2d(input img, flags) → output img”, returning the DFT / IDFT result of the given input. “flags” is a parameter to specify whetherDFT or IDFT is required. You can modify the prototype if necessary.
For the report, please load your input image and use your program to:
1. Perform DFT and manually paste the (centered) Fourier spectrum on your report. (10Points)



2. Perform IDFT on the result of the last question, and paste the real part on your report.Note: the real part should be very similar to your input image. (Why? Think about it.) (5 Points)



3. Detailedly discuss how you implement DFT / IDFT, i.e., the "dft2d" function, in less than2 pages. Please focus on the algorithm part. Don't widely copy/paste your codes in thereport, since your codes are also submitted. (15 Points)

The steps for calculate Fourier transform :

1. centerlize the original image with f(x,y) ×(-1)^(x+y)
2. Apply Fourier transfrom to the image
3. get abs(Spectrum) from the image
4. Apply Fourier log transform.
5. Apply Scalling.

In fact, I use only one function for DFT and IDF because:

$$MNf^*(x,y) = F^*(u,v)$$

```
def dft2d(img, flags):
    if flags == 1:
        return Fdft2d(img)
    elif flags == -1:
        res = np.conjugate(img)
        #return np.conjugate(Fdft2d(img))
        return np.conjugate(Fdft2d(img))
```

At the first, I treated the 2d Fourier transform using 1d method, which is:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-2j\pi(\frac{ux}{M}+\frac{vy}{N})}$$

The time complexity of the transfrom algorithm is O(n^4) which is too slow and I use almost 24 hours to run the program using python. After that I ask a question and I get the answer.

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1} e^{-2j\pi\frac{ux}{M}}\sum_{y=0}^{N-1} f(x,y)e^{-2j\pi\frac{vy}{N}}$$

Because the Fourier tansform is linear and can divided into two steps for implement. First, calculate Fourier transform for cols and then for the rows. Suprisingly and luckly, its time comlexity is O(n^3) which can be done in few minutes.

The core algrothm is:

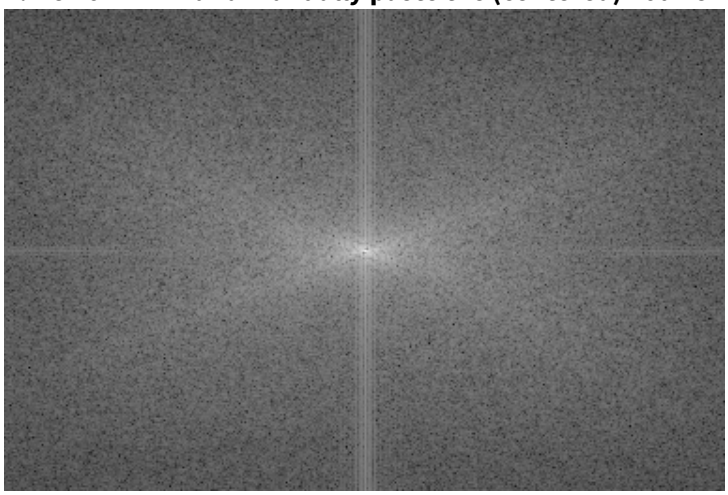
```
    for u in xrange(height):
        for v in xrange(width):
            for y in xrange(width):
                Fourier_img0[u, v] += img[u,y] * (np.exp(-1j*2*np.pi*
(float(v*y)/width)))

    for u in xrange(height):
        for v in xrange(width):
            for x in xrange(height):
                Fourier_img[u, v] += Fourier_img0[x, v] * (np.exp(-1j*2*np.pi*
(float(u*x)/height)))
```

Using the two steps to calculate the whole image.

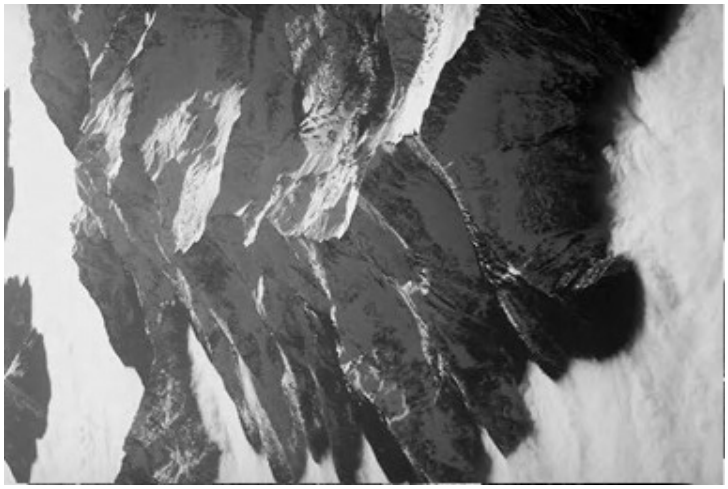
2.3 Bonus: Fast Fourier Transform (50 Points)

This is an optional task. Here you are required to manually implement the Fast Fourier Transform (FFT). The function prototype is "fft2d(input img, flags) → output img", returning the FFT / IFFT result of the given input. "flags" is a parameter to specify whether FFT or IFFT is required. "fft2d" should produce very similar results in comparison to "dft2d". However, your implementation may be limited to images whose sizes are integer powers of 2. We recommend you to handle this problem by simply padding the given input so as to obtain a proper size. For the report, please load your input image and use your program to:

1. Perform FFT and manually paste the (centered) Fourier spectrum on your report. (15 Points)



2. Perform IFFT on the result of the last question, and paste the real part on your report. (10 Points)



3. Detailedly discuss how you implement FFT / IFFT, i.e., the "fft2d" function, in less than 3 pages. (25 Points)

First we have the following thinkings:

1. Inverse Fourier Transform is the same as Fourier Transform.
2. The method in 1d Fourier transfrom can be applied to 2d as well.
3. According to Symmetry, the points in the dialog is at the same value.
4. We have the following equation:

$$
\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,k\,n\,/\,N} \\
&= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i\,2\pi\,k\,(2m)\,/\,N} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i\,2\pi\,k\,(2m+1)\,/\,N} \\
&= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i\,2\pi\,k\,m\,/\,(N/2)} + e^{-i\,2\pi\,k\,/\,N} \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i\,2\pi\,k\,m\,/\,(N/2)}
\end{aligned}
$$

We split a single DFT into two smaller DFT. One odd, one even. So far, we have not save computational overhead, each part contains (n / 2) of N amount of computation, in general, is n.

There is a question "However, your implementation may be limited to images whose sizes are integer powers of 2", and we can simply padding the image to powers of 2. I have two methods for the users:

1. Padding the image to powers of 2.
2. When the program divide the image into some subimage that is not powers of 2, DFT_Slow will be applied.

The core algothmn:

1. First we implentment 1d fft using the thoughts above.

```
def fft(x):
    N = len(x)
    if N <= 1:
        return x
    elif N % 2 != 0:
        return DFT_slow(x)
    else:
        even = fft(x[::2])
        odd = fft(x[1::2])
        return np.concatenate([[even[k] + np.exp(-2j*np.pi*k/N)*odd[k] for k in
```

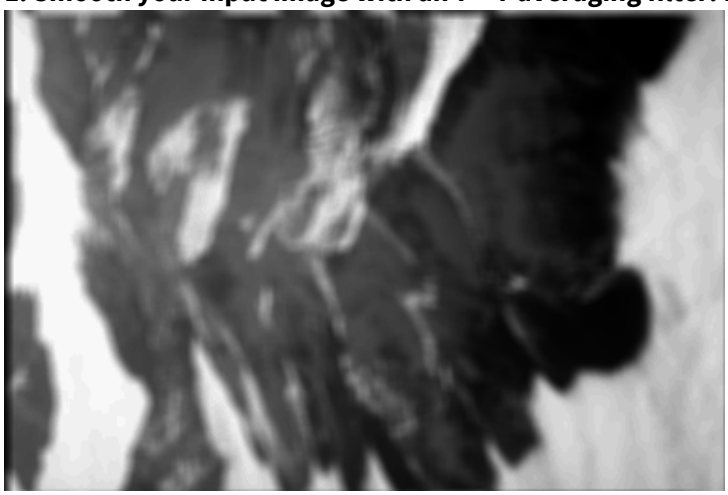
```
    xrange(len(even))]
                ,[even[k] - np.exp(-2j*np.pi*k/N)*odd[k] for k in xrange(len(even))]])
```

1. implentment the 2d fft using 1d fft and using the divide method in fft2d:

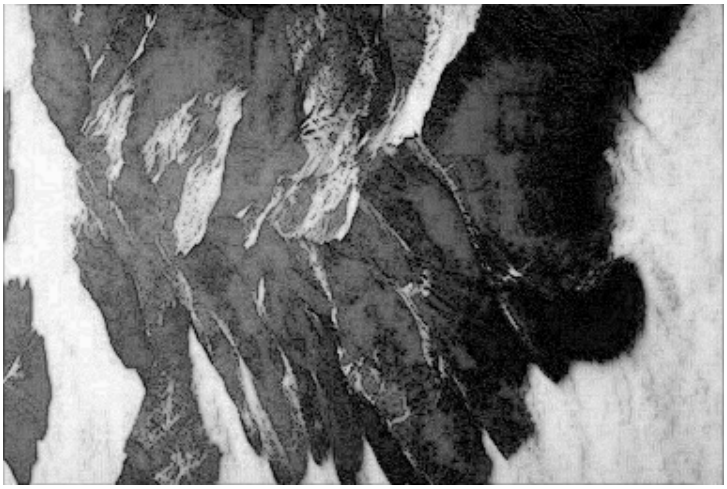
```
def Ffft2d(img):
    height = len(img)
    width = len(img[0])
    Fourier_img0 = np.zeros((height,width),np.complex256)
    for u in xrange(height):
        Fourier_img0[u] = np.array(fft(img[u]))
    Fourier_img = np.zeros((height,width),np.complex256)
    for v in xrange(width):
        Fourier_img[:,v] = fft(Fourier_img0[:,v])

    return Fourier_img
```

As a result, we get the fft2d. And fft is more fast than dft, we can calculate a imgage in seconds rather minutes. For example, dft2d use 3~4 minutes to process the image “14.png”, but fft use only some senconds to process.

2.4 Filtering in the Frequency Domain (30 Points)
Write a function that performs filtering in the frequency domain. The function prototype is “filter2d freq(input img, filter) → output img”, where “filter” is the given filter. Modify the prototype if necessary. According to the convolution theorem, filtering in the frequency domain requires you to apply DFT / FFT to the given image and the given filter, and then multiply them, followed by IDFT / IFFT to get the filtered result. Hence, it should be easy to implement “filter2d freq” based on “dft2d” (or “fft2d”). Of course, you should pay attention to some details like wraparound errors (see Chapter 4.7 of the textbook). For the report, please load your input image and use your “filter2d freq” function to:
1. Smooth your input image with an 7 × 7 averaging filter. Paste your result on the report. (8 Points)



2. Sharpen your input image with a 3 × 3 Laplacian filter and then paste the result (There are four variants of Laplacian in Fig. 3.37 of the textbook. Pick any one you like). (8 Points)



3. Detailedly discuss how you implement the filtering operation, i.e., the "filter2d freq" function, in less than 2 pages. (14 Points)

We can use the following serveral steps to implment the algorithm:

1. Calculate the suitable size for the image which is suppose to be powers 2 and then padding zeros for both the origin image and the filter simply puting them at the letf top conrner. (Because in frequency domain, it does not matter where you put the image, but will have some side effect for the image's edge).
   Resize function:

```
def calculate_pow2(x):
    res = 2
    while res < x:
        res = res << 1
    return res
```

2. Apply Fourier transform for both the images. Centerlize them respetively and then apply fft2d in question2(because it's fast than using fft2d).

```
FFT_filter = FFT.centralize(FFT_filter)
adjust_img = FFT.centralize(adjust_img)

FFT_filter = FFT.fft2d(FFT_filter, 1)
FFT_image = FFT.fft2d(adjust_img, 1)
```

3. Using Convolution theorem:

$$f(x,y) * h(x,y) \Leftarrow FT \Rightarrow F(u,v) \cdot H(x,y)$$

   Let the two matrix to do array multiplication. And get the result image.

```
Result = FFT_image * FFT_filter
Result = FFT.fft2d(Result, -1).real
```

4. Apply Inverse Fourier transform for the image, and get the real part as well as apply centerlize again.

5. Return the iamge and then do scaling for display.

Deatils：

1. For laplacian filtering, the formula should be applied like that in project 2.:

   Result[x, y] = 5 * img[x, y] - Result[x, y]

2. The output image will become brighter. In this way, I apply histogram equlization in project 2. This time I use gama correction.

3. Log scaling is more better than linear:

$$f'(x,y) = \frac{255}{log_{10}256} \times log_2(1 + \frac{|f(x,y)|}{max(f(x,y))} \times 255)$$

Note:
1. Your results of the first two questions should be very similar to the corresponding results in HW2 (if your previous implementation for HW2 is correct).