

# DIP Homework 3

Qiuyi Zhang 12330402  
joyeec9h3@gmail.com

December 2, 2014

## Contents

<b>1 Exercises</b>	<b>1</b>
1.1 Rotation . . . . .	1
1.2 Translation . . . . .	1
1.3 Filtering in the Frequency Domain . . . . .	2
1.4 Highpass Filter . . . . .	2
<b>2 Programming Tasks</b>	<b>3</b>
2.1 Fourier Transform . . . . .	3
2.1.1 Results . . . . .	3
2.1.2 Discussion . . . . .	4
2.1.3 Algorithm . . . . .	5
2.2 Fast Fourier Transform . . . . .	5
2.2.1 Results . . . . .	5
2.2.2 Discussion . . . . .	6
2.2.3 Algorithm . . . . .	7
2.3 Filtering in the Frequency Domain . . . . .	7
2.3.1 Results . . . . .	7
2.3.2 Discussion . . . . .	9
2.3.3 Algorithm . . . . .	9

## 1 Exercises

### 1.1 Rotation

**Answer:**

Taking the complex conjugate will change  $j2\pi$  to  $-j2\pi$  in the inverse transform formula,

$$\begin{aligned}\mathfrak{F}^{-1}[F^*(u, v)] &= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{-j2\pi(ux/M+vy/N)} \\ &= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(u(-x)/M+v(-y)/N)} \\ &= f(-x, -y)\end{aligned}$$

Therefore step 3 will effectively mirror  $f(x, y)$  about the origin, producing Fig. 1(b).

### 1.2 Translation

**Answer:**

$$\begin{aligned}
F(u, v)e^{-j2\pi(\frac{ux_0}{M} + \frac{vy_0}{N})} &= \frac{1}{MN}e^{-j2\pi(\frac{ux_0}{M} + \frac{vy_0}{N})} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f(x, y)e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})} \\
&= \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f(x, y)e^{j2\pi(\frac{u(x-x_0)}{M} + \frac{v(y-y_0)}{N})} \\
&= f(x - x_0, y - y_0)
\end{aligned}$$

### 1.3 Filtering in the Frequency Domain

**Answer:**

Since the borders of the original image are not black, padding the image with zeroes introduces sharp transitions at the borders in the spatial domain, which corresponds to the high-frequency components along the vertical and horizontal directions in the frequency domain. Hence the signal along the vertical and horizontal axes are significantly strengthened.

### 1.4 Highpass Filter

**Answer:**

$$\begin{aligned}
h(x, y) &= 2f(x, y) - f(x + 1, y) - f(x - 1, y) \\
g(x, y) &= 3f(x, y) - f(x + 1, y) - f(x - 1, y)
\end{aligned}$$

Using the translation property:

$$\begin{aligned}
G(u, v) &= \mathfrak{F}[g(x, y)] \\
&= (3 - e^{\frac{2j\pi u}{M}} - e^{\frac{-2j\pi u}{M}})F(u, v) \\
&= H(u, v)F(u, v) \\
H(u, v) &= 3 - e^{\frac{2j\pi u}{M}} - e^{\frac{-2j\pi u}{M}} \\
&= 3 - (\cos \frac{2\pi u}{M} + j \sin \frac{2\pi u}{M} + \cos \frac{-2\pi u}{M} + j \sin \frac{-2\pi u}{M}) \\
&= 3 - (\cos \frac{2\pi u}{M} + j \sin \frac{2\pi u}{M} + \cos \frac{2\pi u}{M} - j \sin \frac{2\pi u}{M}) \\
&= 3 - 2 \cos \frac{2\pi u}{M}
\end{aligned}$$

The centered form of  $H(u, v)$  is

$$3 - 2 \cos \frac{2\pi(u - \frac{M}{2})}{M}$$

As  $u$  ranges from 0 to  $M-1$ , the value of  $H(u, v)$  starts at 5, falls to 1 when  $u = \frac{M}{2}$  (the center of the filter) and then climbs to 5 again when  $u = M$ , as shown in Figure 1. Therefore, the amplitude of the filter increases as a function of distance from the origin of the centered filter, which is the characteristic of a highpass filter.

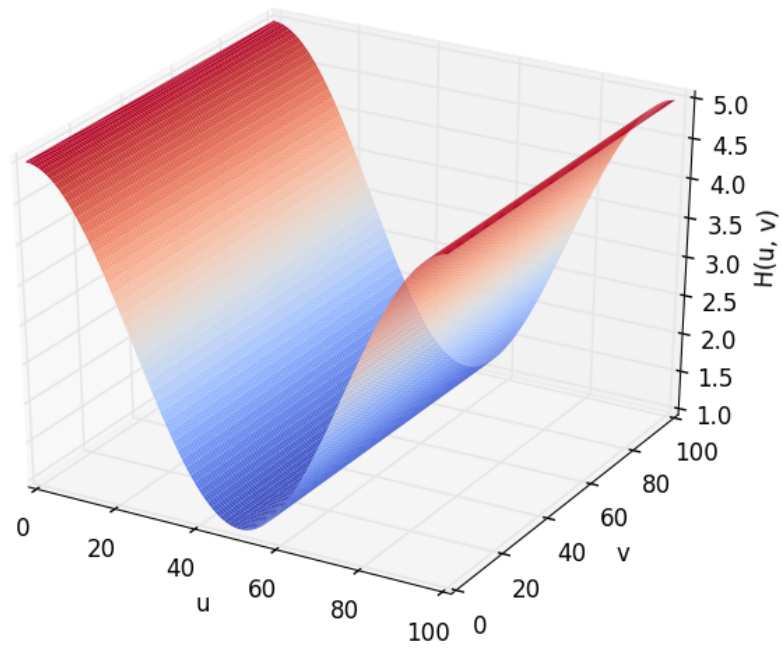


Figure 1:  $H(u, v)$  where  $M = N = 100$

## 2 Programming Tasks

### 2.1 Fourier Transform

#### 2.1.1 Results



Figure 2: The original image

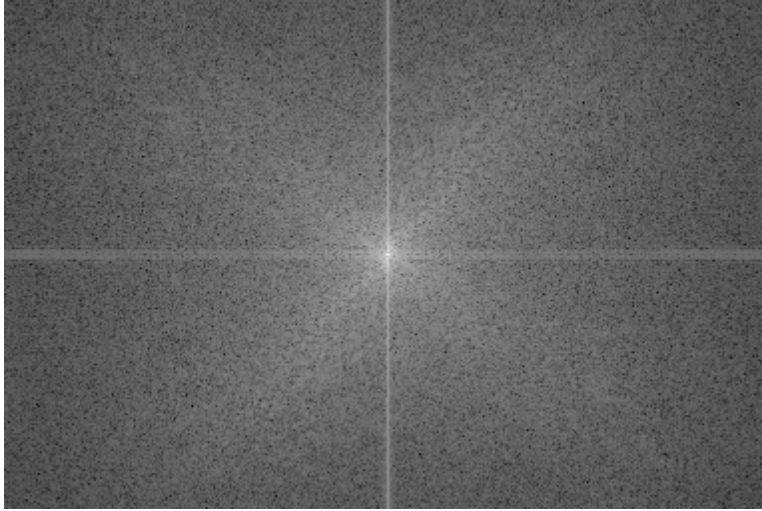


Figure 3: The centered Fourier spectrum



Figure 4: The real part of the IDFT of the DFT

### 2.1.2 Discussion

Since the data obtained from the original image is real, and

$$\mathfrak{F}^{-1}[\mathfrak{F}[f(x, y)]] \equiv f(x, y)$$

if we take the DFT of the original image, then apply IDFT, theoretically the real part of the result is the same as the original image. In practice, however, there might be some small errors caused by the limited precision of floating numbers.

To accelerate the calculation, we can pre-compute the discrete fourier transform matrix. Let  $N$  be the size of the 1D vector  $\mathbf{f}$  that will be transformed, the discrete fourier transform matrix is:

$$\mathbf{W}_{x,y} = e^{\frac{j2\pi xy}{N}}$$

Then the discrete fourier transform  $\mathbf{F} = \mathbf{W}\mathbf{f}$ , that is, the dot product of the transform matrix and the original vector. Using the separability of the fourier transform, for a  $M \times N$  matrix  $f$ , its discrete fourier transform  $\mathbf{F} = \mathbf{W}_M \mathbf{f} \mathbf{W}_N$ , where  $\mathbf{W}_M$  and  $\mathbf{W}_N$  are discrete fourier transform matrices for vectors of size  $M$  and  $N$ .

Similarly, the transform matrix for discrete inverse fourier transform is:

$$\mathbf{W}_{x,y}^i = e^{\frac{-j2\pi xy}{N}}$$

The 1D transform is  $f = \frac{1}{N} \mathbf{W}^i \mathbf{F}$ , the 2D transform is  $f = \frac{1}{MN} \mathbf{W}_M^i \mathbf{F} \mathbf{W}_N^i$

### 2.1.3 Algorithm

---

#### Algorithm 1 Discrete Fourier Transform

---

```

1: function DFT2D(input_img, flags)
2:    $M$  = number of rows in input_img
3:    $N$  = number of columns in input_img
4:   if flags = 1 then
5:      $\mathbf{W}_M = \text{DFTMTX}(M)$ 
6:      $\mathbf{W}_N = \text{DFTMTX}(N)$ 
7:     return  $\mathbf{W}_M \mathbf{f} \mathbf{W}_N$ 
8:   else
9:      $\mathbf{W}_M^i = \text{IDFTMTX}(M)$ 
10:     $\mathbf{W}_N^i = \text{IDFTMTX}(N)$ 
11:    return  $\frac{1}{MN} \mathbf{W}_M^i \mathbf{f}^i \mathbf{W}_N^i$ 
12:   end if
13: end function
14:
15: function DFTMTX( $N$ )
16:   return Matrix  $\mathbf{W}_{x,y} = e^{\frac{j2\pi xy}{N}}$ 
17: end function
18:
19: function IDFTMTX( $N$ )
20:   return Matrix  $\mathbf{W}_{i,x,y} = e^{\frac{-j2\pi xy}{N}}$ 
21: end function

```

---

## 2.2 Fast Fourier Transform

### 2.2.1 Results



Figure 5: The original image

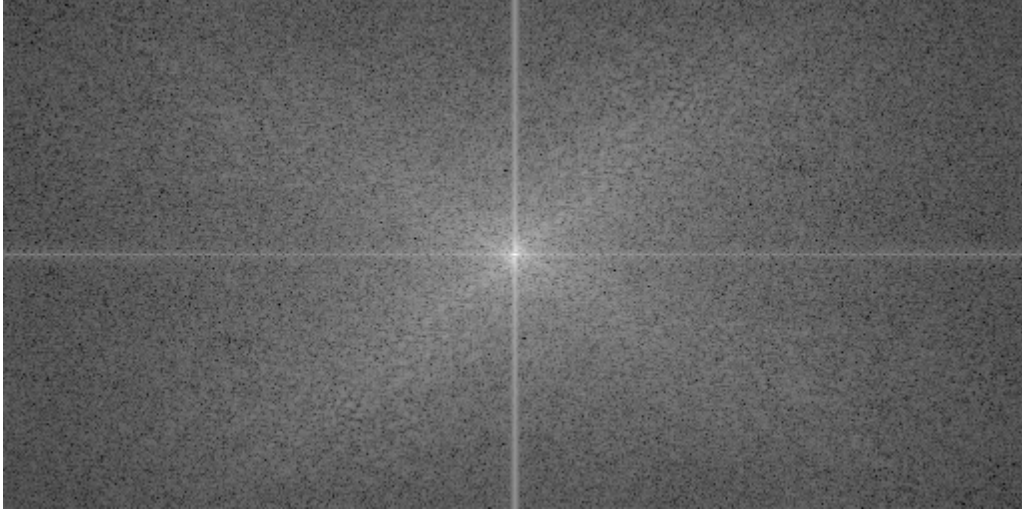


Figure 6: The centered Fourier spectrum



Figure 7: The real part of the IDFT of the DFT

### 2.2.2 Discussion

Here I use the CooleyTukey algorithm to compute the fast fourier transform, which requires the length of the input vector to be power of 2.

The recursive version is easy to implement, but it takes a bit too long. To accelerate it, we can rewrite it into a iterative version. Since in each step, the odd parts in subproblems are actually the latter half of all subproblems of the previous step, and the even parts the former half, we can concatenate all odd parts and all even parts, and use the symmetry of the twiddle factor to vectorize the butterfly operation.

In addition, when the length of the input is smaller than a cut-off value (on my computer, 16), then it could be faster to use the vectorized DFT implemented before. We can use it to optimize for our smaller subproblems.

Because of the seperability of DFT, The 2D FFT can be obtained by simply applying the 1D FFT over each row of the matrix, and then apply it over each column.

Since

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

to implement the IFFT, we can compute the 2D forward FFT for the complex conjugate of the data, divide the result by  $MN$ , and then take the complex conjugate again.

### 2.2.3 Algorithm

---

#### Algorithm 2 2D Fast Fourier Transform

---

```

1: function FFT2D(input_img, flags)
2:   if flags == 1 then
3:     Apply FFT over each row of input_img
4:     Apply FFT over each column of input_img
5:     return The transformed matrix
6:   else
7:     output_img = the complex conjugate of input_img
8:     output_img = FFT2D(input_img, -1)
9:     return the complex conjugate of output_img
10:  end if
11: end function

```

---



---

#### Algorithm 3 1D Fast Fourier Transform(recursive)

---

```

1: function FFT(x)
2:   if The length of x == 1 then return x
3:   else
4:     even = elements of x with even indices
5:     odd = elements of x with even indices
6:     even = FFT(even)
7:     odd = FFT(odd)
8:     coff =  $\mathbf{W}_x = e^{\frac{j2\pi x}{N}}$ 
9:     first = even + coff * odd
10:    second = even - coff * odd
11:    return concatenation of first and second
12:  end if
13: end function

```

---



---

#### Algorithm 4 1D Fast Fourier Transform (iterative)

---

```

1: function FFT(x)
2:   while x is not a  $N \times 1$  vector do
3:     m = number of rows in x
4:     even = first half of columns of x
5:     odd = second half of columns of x
6:     coff =  $\mathbf{W}_x = e^{\frac{j\pi x}{N}}$  ▷ Because of the concatenation,  $m = 2N$ , cancels out 2
7:     twiddle = mutiply each column of odd by coff
8:     x = concatenation of even + twiddle and even - twiddle
9:   end while
10:  return x
11: end function

```

---

## 2.3 Filtering in the Frequency Domain

### 2.3.1 Results

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 8: Laplacian filter



Figure 9: The original image

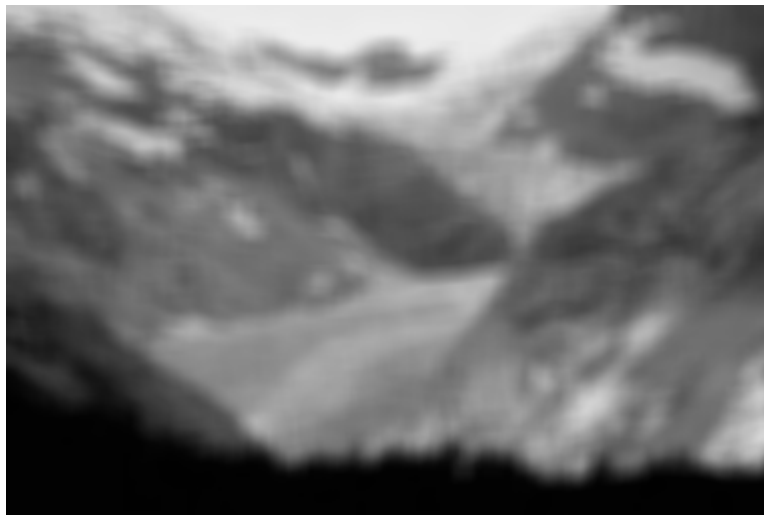


Figure 10: Smoothed image with  $11 \times 11$  averaging filter

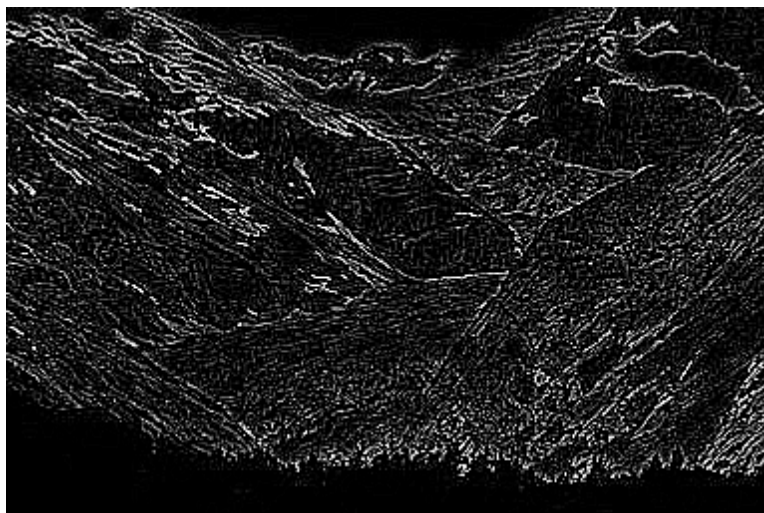


Figure 11: Image filtered with  $3 \times 3$  Laplacian filter



### Note

To obtain a sharpened image with the result of a laplacian filter, we can combine the original image with the filtered result. Because the laplacian filter used here has a positive center, the sharpened image can be produced by:

$$g(x, y) = f(x, y) + \text{laplacian}(x, y)$$

where  $g(x, y)$ ,  $f(x, y)$ , and  $\text{laplacian}(x, y)$  are intensity values at the pixels of coordinate  $(x, y)$  in the sharpened image, original image, and the laplacian filtered image, respectively.

### 2.3.2 Discussion

Since in the implementation, the array storing the input of the fourier transform doesn't support negative indices, the filter is actually translated by  $(\frac{m}{2}, \frac{n}{2})$ . Therefore the result needs to be cropped back after reversing this translation.

### 2.3.3 Algorithm

#### Note

I use the duplicate of borders to pad the input before filtering, and then crop out the result to avoid the inconsistency around the borders.

---

#### Algorithm 5 Filter

---

```
1: function FILTER2D_FREQ(input_img, filter)
2:    $M, N$  = height and width of input_img
3:    $m, n$  = height and width of filter
4:    $P, Q$  = the nearest larger power of 2 of  $M + m - 1$  and  $N + n - 1$ 
5:    $f_p$  = input_img padded to the upper left corner of a  $P \times Q$  zero matrix
6:    $h_p$  = filter padded to the upper left corner of a  $P \times Q$  zero matrix
7:    $F_{uv} = \text{DFT2D}(f_p, 1)$ 
8:    $H_{uv} = \text{DFT2D}(h_p, 1)$ 
9:    $G_{uv} = F_{uv}H_{uv}$ 
10:  result = The real part of  $\text{DFT2D}(G_{uv}, -1)$ 
11:  return The image cropped from result, starting from  $(\frac{m}{2}, \frac{n}{2})$ 
12: end function
```

---