DIP Homework 3

Qiuyi Zhang 12330402
joyeec9h3@gmail.com

December 2, 2014

Contents

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:

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

Using the translation property:

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

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

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

As u ranges from 0 to $M1$, the value of $H(u,v)$ starts at 4, falls to 1 when $u = \frac{M}{2}$ (the center of the filter) and then climbs to 4 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.

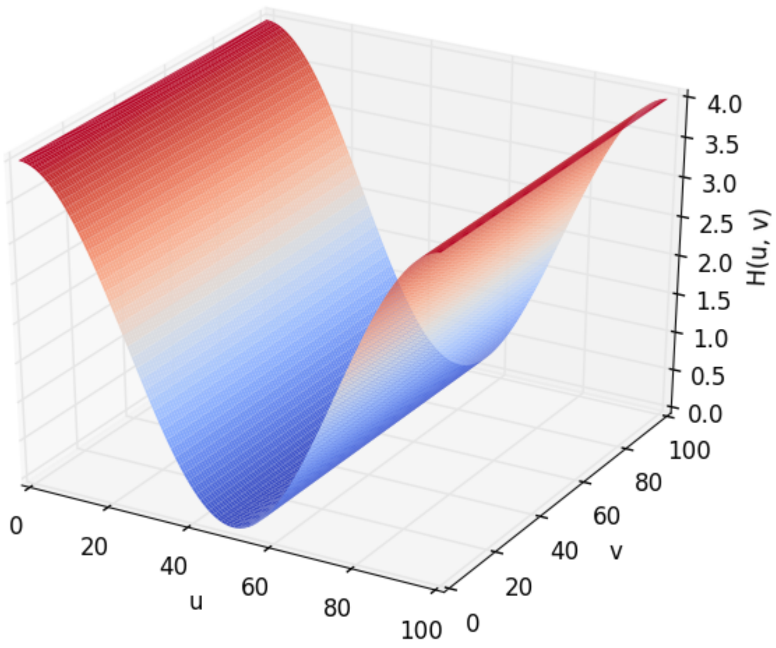

Figure 1: $H(u, v)$ where $M = N = 100$

2 Programming Tasks

2.1 Fourier Transform

2.1.1 Results

Figure 2: The original image

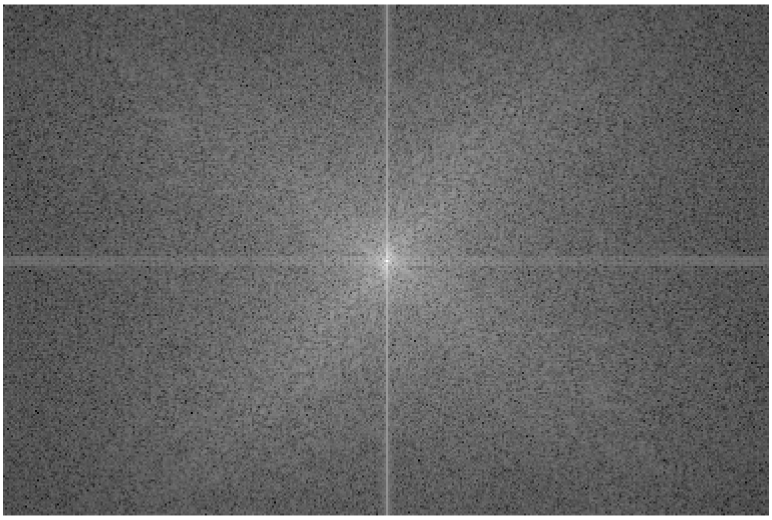

Figure 3: The centered Fourier spectrum

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 presicion 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:

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

The 1D transform is $f = \frac{1}{N}\mathbf{W^iF}$, the 2D transform is $f = \frac{1}{MN}\mathbf{W^i_M F W^i_N}$

2.1.3 Algorithm

Algorithm 1 Discrete Fourier Transform

function DFT2D(*input_img*, flags)
    M = number of rows in *input_img*
    N = number of columns in *input_img*
    if flags = 1 **then**
        $\mathbf{W_M}$ = DFTMTX(M)
        $\mathbf{W_N}$ = DFTMTX(N)
        return $\mathbf{W_M f W_N}$
    else
        $\mathbf{W^i_M}$ = IDFTMTX(M)
        $\mathbf{W^i_N}$ = IDFTMTX(N)
        return $\frac{1}{MN}\mathbf{W^i_M f^i W_N}$
    end if
end function

function DFTMTX(N)
    return Matrix $\mathbf{W}_{x,y} = e^{\frac{j2\pi xy}{N}}$
end function

function IDFTMTX(N)
    return Matrix $\mathbf{W_i}_{x,y} = e^{\frac{-j2\pi xy}{N}}$
end function

2.2 Fast Fourier Transform

2.2.1 Results

Figure 5: The original image

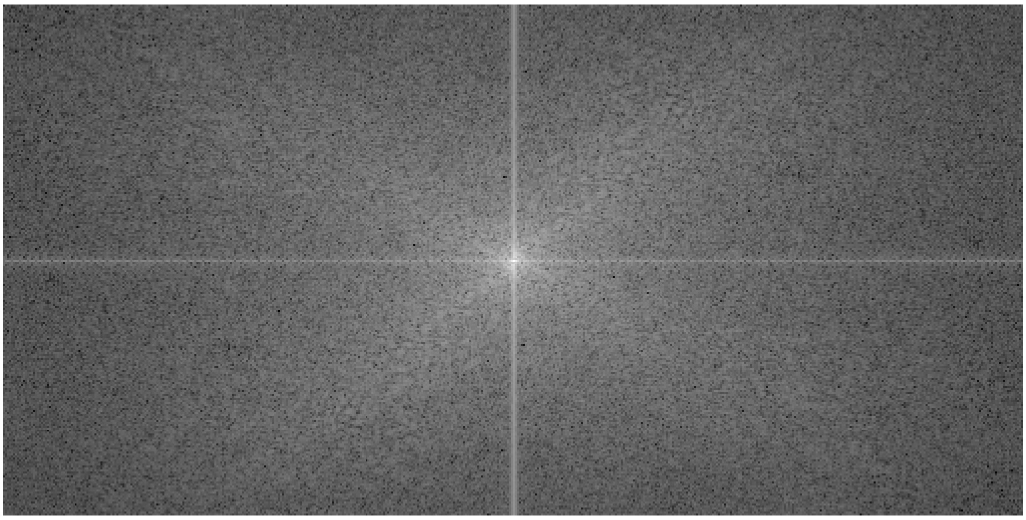

Figure 6: The centered Fourier spectrum

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

function FFT2D(*input_img*, flags)
  if flags == 1 **then**
    Apply FFT over each row of *input_img*
    Apply FFT over each column of *input_img*
    return The transformed matrix
  else
    output_img = the complex conjugate of *input_img*
    output_img = FFT2D(*input_img*, -1)
    return the complex conjugate of *output_img*
  end if
end function

Algorithm 3 1D Fast Fourier Transform(recursive)

function FFT(x)
  if The length of x == 1 **then return** x
  else
    $even$ = elements of x with even indices
    odd = elements of x with even indices
    $even$ = FFT($even$)
    odd = FFT(odd)
    $coff = \mathbf{W}_x = e^{\frac{j2\pi x}{N}}$
    $first = even + coff * odd$
    $second = even$ - $coff * odd$
    return concatenation of $first$ and $second$
  end if
end function

Algorithm 4 1D Fast Fourier Transform (iterative)

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

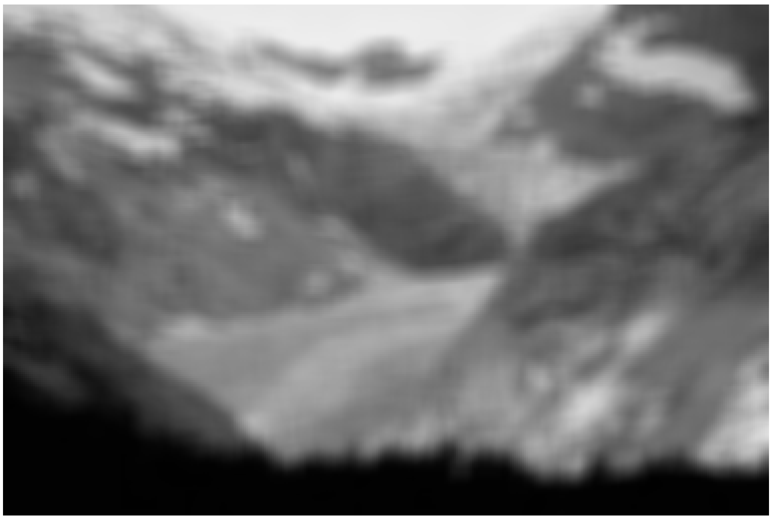

Figure 10: Smoothed image with 11×11 averaging filter

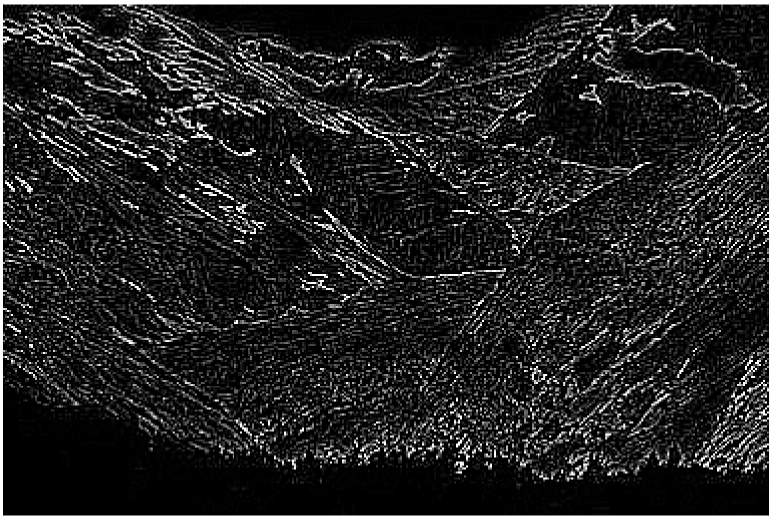

Figure 11: Image filtered with 3×3 Laplacian filter

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:

$$g(x, y) = f(x, y) + laplacian(x, y)$$

where $g(x, y)$, $f(x, y)$, and $laplacian(x, y)$ 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

```
function FILTER2D_FREQ(input_img, filter)
    M, N = height and width of input_img
    m, n = height and width of filter
    P, Q = the nearest larger power of 2 of M + m − 1 and N + n − 1
    f_p = input_img padded to the upper left corner of a P × Q zero matrix
    h_p = filter padded to the upper left corner of a P × Q zero matrix
    F_uv = DFT2D(f_p, 1)
    H_uv = DFT2D(h_p, 1)
    G_uv = F_uv H_uv
    result = The real part of DFT2D(G_uv, −1)
    return The image cropped from result, starting from (m/2, n/2)
end function
```