Signal and image processing - Assignment 3

Fengmao Wang(mnp553), Ji Wu (tlq412), Bo Cui(wvm400)

February 28, 2022

1 Fourier Transform – Theory

1.1

Fourier series is an expansion of periodic signal as a linear combination of sines and cosines while Fourier transform is the process or function used to convert signals from time domain in to frequency domain.The Fourier series is used to represent a periodic function by a discrete sum of complex exponentials, while the Fourier transform is then used to represent a general, nonperiodic function by a continuous superposition or integral of complex exponentials. The Fourier transform can be viewed as the limit of the Fourier series of a function with the period approaches to infinity, so the limits of integration change from one period to $(-\infty, \infty)$.

1.2

proof

We define $\epsilon(x)$ as our real and even function, so we get the derivation for its Fourier transform as follow,

$$\begin{aligned}\mathcal{F}(\epsilon(x)) &= \int_{-\infty}^{\infty}(\epsilon(x))e^{-i2\pi ux}dx \\ &= \int_{-\infty}^{\infty}(\epsilon(x))(cos(2\pi ux) - isin(2\pi ux))dx \quad \text{(use the basis in slide 4 of Fourier1)} \\ &= \int_{-\infty}^{\infty}\epsilon(x)cos(2\pi ux) - i\int_{-\infty}^{\infty}\epsilon(x)sin(2\pi ux)dx\end{aligned}$$

Since $sin(2\pi ux)$ is odd function and $\epsilon(x)$ is even function, then their product $\epsilon(x)\cdot sin(2\pi ux)$ is also a odd function. So its integral $\int_{-\infty}^{\infty}\epsilon(x)sin(2\pi ux)dx = 0$. Therefore, the Fourier transform is real.

So we have,

$$\begin{aligned} F(u) &= \mathcal{F}(\epsilon(x)) \\ &= \int_{-\infty}^{\infty} \epsilon(x) cos(2\pi ux) dx \\ &= \int_{-\infty}^{\infty} \epsilon(x) cos(-2\pi ux) dx \quad \text{(even function } cos(2\pi ux) = cos(-2\pi ux)) \\ &= F(-u) \end{aligned}$$

Hence, the continuous Fourier transform of a real and even function is real and even.

1.3

The Derivation of its Fourier transform is as follow,

$$\begin{aligned} \mathcal{F}(\delta(x-d)+\delta(x+d)) &= \int_{-\infty}^{\infty} (\delta(x-d)+\delta(x+d)) e^{-i2\pi ux} dx \\ &= \int_{-\infty}^{\infty} \delta(x-d) e^{-i2\pi ux} dx + \int_{-\infty}^{\infty} \delta(x+d) e^{-i2\pi ux} dx \\ &= \int_{-\infty}^{\infty} \delta(z) e^{-i2\pi u(z+d)} dz + \int_{-\infty}^{\infty} \delta(z) e^{-i2\pi u(z-d)} dz \quad \text{(change of variables)} \\ &= e^{-i2\pi ud} \int_{-\infty}^{\infty} \delta(z) e^{-i2\pi uz} dz + e^{i2\pi ud} \int_{-\infty}^{\infty} \delta(z) e^{-i2\pi uz} dz \\ &= e^{-i2\pi ud} \mathcal{F}(\delta(x)) + e^{i2\pi ud} \mathcal{F}(\delta(x)) \quad \text{(reordering)} \\ &= e^{-i2\pi ud} + e^{i2\pi ud} \quad (\mathcal{F}(\delta(x)) = 1) \end{aligned}$$

1.4

1.4.1

According to the definition of integral, we have

$$\begin{aligned} \int_{-\infty}^{\infty} b_a(x) dx &= \int_{-\frac{a}{2}}^{\frac{a}{2}} \frac{1}{a} dx \\ &= \left[\frac{x}{a}\right]_{-\frac{a}{2}}^{\frac{a}{2}} \\ &= \frac{1}{a}\left[\frac{a}{2} - (-\frac{a}{2})\right] \\ &= 1 \end{aligned}$$

1.4.2

Using the definition of the Fourier transform, we have

$$\begin{aligned}
\mathcal{F}(s) &= \int_{-\frac{a}{2}}^{\frac{a}{2}} \frac{1}{a} e^{-i2\pi sx} dx \\
&= \int_{-\frac{a}{2}}^{\frac{a}{2}} \frac{1}{a} \cos(2\pi sx) dx - \int_{-\frac{a}{2}}^{\frac{a}{2}} \frac{1}{a} i \sin(2\pi sx) dx \\
&= \frac{1}{2\pi sa} \left[\sin(2\pi sx)\right]_{-\frac{a}{2}}^{\frac{a}{2}} + 0 \\
&= \frac{\sin(\pi sa)}{\pi sa} \\
&= \operatorname{sinc}(\pi sa)
\end{aligned}$$

Thus, $B_a(k)$ can be written as $B_a(k) = \operatorname{sinc}(ak\pi)$

1.4.3

Because $\lim_{x\to 0} \frac{\sin x}{x} = 1$,

$$\begin{aligned}
\lim_{a\to 0} B_a(k) &= \lim_{a\to 0} \frac{\sin(ak\pi)}{ak\pi} \text{ (when } a \to 0,\ ak\pi \to 0) \\
&= 1
\end{aligned}$$

1.4.4

As a is near zero, its Fourier transform is wide in the frequency domain. The wider the time domain, the narrower the frequency domain and vice versa.

$$\begin{aligned}
f(t) &\leftrightarrow F(s) \\
f(at) &\leftrightarrow \frac{1}{|a|} F(\frac{s}{a})
\end{aligned}$$

It can be seen from this formula that the transformation relationship between the frequency domain and the time domain is opposite.
Observing the form of Fourier transform of $B_a(k)$,

$$B_a(k) = \operatorname{sinc}(ak\pi) = \frac{\sin(ak\pi)}{ak\pi}$$

So we know that the period of $\sin(ak\pi)$ is $2\pi/a\pi = \frac{2}{a}$.
When a is near zero, the period of $\sin(ak\pi) \to \infty$, which means for a period, its x-axis is infinitely stretched. Apart from that, for the denominator $ak\pi$, it just adapts to the value of a after the change of a, namely, for example, for $a' = 2a$, then we have the same value for $B_a(2k) = B_{a'}(k)$. So the change of a does not effect the values in y-axis.
Hence, its Fourier transform is wider in the frequency domain when a is near zero.
In the contrary, when a is large, its Fourier transform is narrow in the frequency domain.

We can see the trend as well by comparing the graphs as follow.

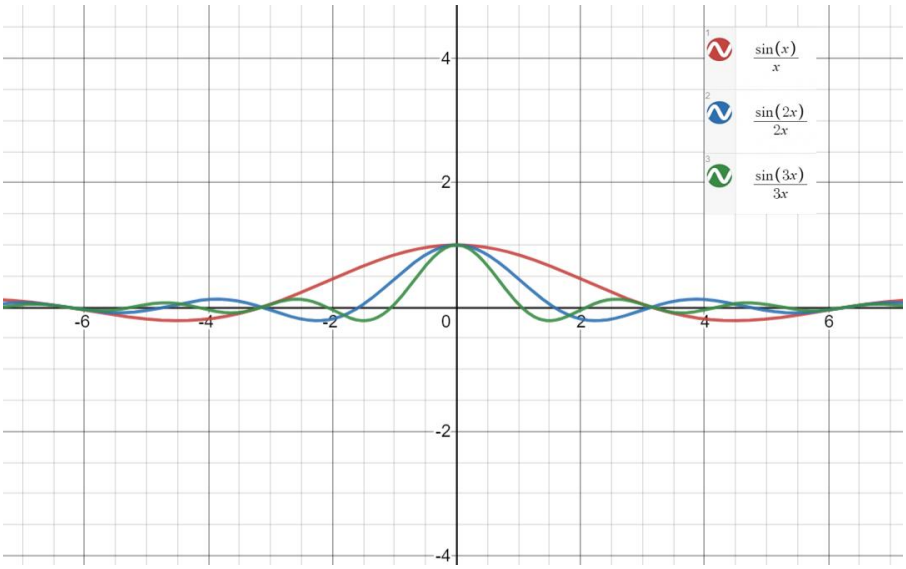

Figure 1: the graph of $\frac{sin(x)}{x}$, $\frac{sin(2x)}{2x}$ and $\frac{sin(3x)}{3x}$

2 Fourier Transform – Practice

2.1

We use Python to calculate the power spectrum of trui.png. The result can be seen in Figure 2. The essential code is included in Listing 1. We know that the power spectrum is the square of the magnitude, so I use magnitude to calculate power spectrum. And use log to display the result. We can see that most of the energy is concentrated at low frequencies.

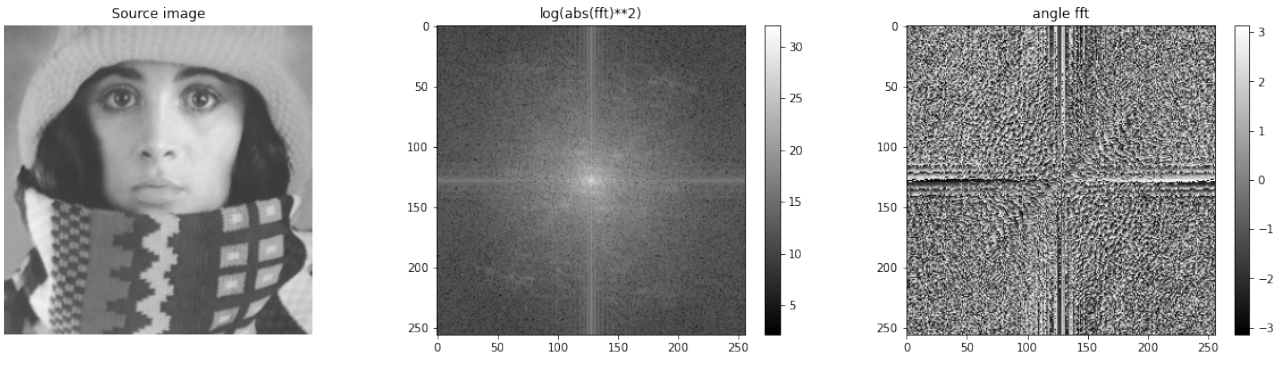

Figure 2: Original image of 'trui.png' and its power spectrum

```python
def image_fft(image):
  fft_img = fft.fft2(image)
  fft_shift_img = fft.fftshift(fft_img)
  magnitude = np.abs(fft_shift_img)
  phase = np.angle(fft_shift_img)
  return magnitude, phase
```

Listing 1: Code for calculating the power spectrum

2.2

We implemented convolution in two ways. The first is nested for loop and the second is using fast Fourier transformation. The essential code is included in Listing 2. A example could be seen in Figure 3, we calculated the convolution of the image 'trui.png' with kernel = np.ones([5,5]). We found that the we got basically same results from two different convolution methods. And the time of FFT is increasing very slowly.

```python
def convolution2d(image, kernel):
    w, h = image.shape
    kernel = np.flip(kernel)
    m, n = kernel.shape
    kernel_w = m // 2
    kernel_h = n // 2
    # padding process for boundary pixels
    _img = np.zeros((w+2*kernel_w,h+2*kernel_h), dtype=float)
    _img[kernel_w:kernel_w+w,kernel_h:kernel_h+h] = image.copy()
    padding_img = _img.copy()
    new_image = np.zeros_like(padding_img)
    for i in range(kernel_w, kernel_w + w):
        for j in range(kernel_h, kernel_h + h):
            window = padding_img[i-kernel_w : int(np.ceil(i+m/2)),
                                 j-kernel_h : int(np.ceil(j+n/2))]
            new_image[i, j] = np.sum(window * kernel)
    return new_image[kernel_w : w+kernel_w , kernel_h : h+kernel_h]


def convolution_fft(image, kernel):
    fft_image = fft.fft2(image)
    fft_kernel = fft.fft2(kernel, image.shape)
    fft_new_image = fft_image * fft_kernel
    new_image = fft.ifft2(fft_new_image, image.shape)
    return new_image
```

Listing 2: Code for convolution in For Loop and FFT

Figure 3: Original image of 'trui.png' and its convolutions with kernel = np.ones([5,5])

To compare the two implementations. In Figure 4, we recorded the computation time of the convolution of image 'trui.png' and a number of kernel sizes. And Figure 5 is about the computation time of the convolution of fixed kernel and many image sizes. We could see that with the increasing of kernel size or image size, the time of for loop is increasing much faster than FFT. And only when both kernel and image having small sizes, for loop may take less time than FFT.

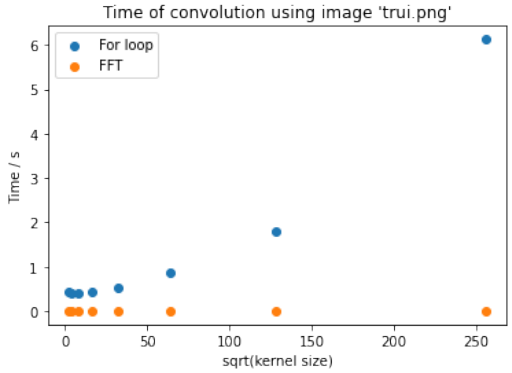

Figure 4: Computation time of image 'trui.png' and a number of kernel sizes

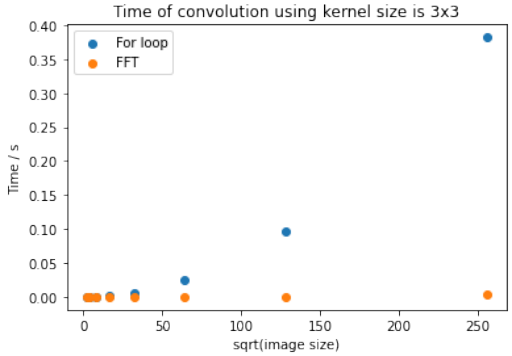

Figure 5: Computation time of a fixed kernel size(3x3) and different image sizes

2.3

We implemented the function for adding planar waves $a_0 cos(v_0 x + w_0 y)$ to a image and also designed a filter removing any such planar waves given v_0 and w_0. The essential code is included in Listing 3. We designed a filter that could remove the certain frequency component from frequency domain. Since we know that planar waves in frequency domain are two points with infinite value, we could use this property to design a filter by removing those two points in frequency domain. But that is ideal situation, because for digital signal we could do FFT which is a kind of DFT and DFT can be seen as the sampled version of the DTFT output in frequency-domain. So we cannot find two points with infinite value from the output of planar wave's FFT, because it lost information and accuracy in sampling.

So for the filter, I will do FFT of planar wave to obtain the magnitude in frequency-domain and normalize the log of magnitude. After normalization, I will regard any points with value greater than 0.5 as the frequency component of that planar wave. Then use this as a mask which means greater than 0.5 is one, otherwise is zero. And do FFT of input image and multiply the result with the mask. Finally do IFFT of product and return this as output which has removed such planar waves.

```
def cos2d(x1, x2, alpha, u, v):
    return alpha*np.cos(u*x1 + v*x2)

def add_planar_waves(image, alpha, u, v):
    m, n = image.shape
    M = np.linspace(0, m, m)
    N = np.linspace(0, n, n)
    X1, X2 = np.meshgrid(M, N)
    planar_waves = cos2d(X1, X2, alpha, u, v)
    new_image = image.astype(float) + planar_waves
    return new_image

def filter_planar_waves(image, u, v):
    m, n = image.shape
    fft_image = fft.fft2(image)

    blank_image = np.zeros([m, n])
    planar_waves = add_planar_waves(blank_image, 1, u, v)
    fft_planar_waves = fft.fft2(planar_waves)

    mask = np.log(abs(fft_planar_waves)) / np.max(np.max(np.log(abs(fft_planar_waves)))
    mask = np.where(mask > 0.5, 0, 1)
    fft_new_image = fft_image * mask
    new_image = fft.ifft2(fft_new_image, image.shape)
    return new_image
```

Listing 3: Code for adding planar waves to image and the filter

In Figure 6, we could see the result of adding planar waves and filtering. In the power spectrum of image adding planar waves, we could see two light points which are the frequency component of planar waves. And after filtering, we could see that the those points were removed and having very low values.

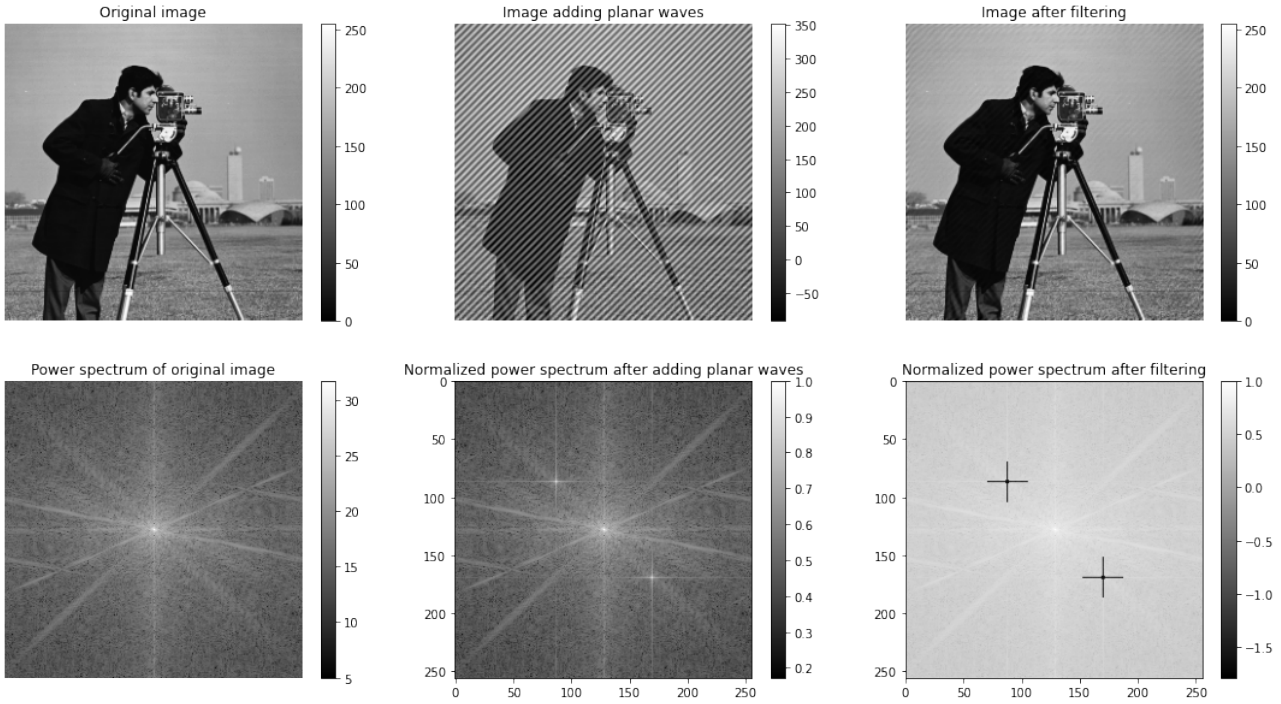

Figure 6: Original image of cameraman.tif and image after adding planar waves ($100\cos(x+y)$) and after filtering, with corresponding power spectrums