C032 - Fourier Optics

Lucy Symes Thompson - Wadham College

20/05/2017

1 Abstract

In this exercise I observe the results of a 1D discrete Fourier transform for a single slit, varying both the position and transmittivity of the slit. I also verify the convolution theorem. Similarly, I observe the results of a 2D discrete Fourier transform for a cross-shape and two square-shaped sources.

2 Introduction

Fourier methods are used a lot in optics as each electromagnetic wave can be considered a superposition of plane waves; these are found using Fourier analysis. In practice, a given aperture function is decomposed into k-components, and these represent the transmitted light. Sometimes the integral can be solved analytically; other times it has to be discretised and solved numerically. We will be looking at the discrete Fourier transform (DFT) and its efficacy.

3 Theoretical Overview

The continuous 1D Fourier transform is given as:

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{i2\pi kx} dx \tag{1}$$

And its inverse is given as:

$$f(x) = \int_{-\infty}^{\infty} F(k) e^{-i2\pi kx} dk \tag{2}$$

In this case, the aperture function is f(x).

We can approximate this integral by discretising it as follows:

$$F(k) = \frac{1}{2N} \sum_{x=-N}^{N-1} f(x) e^{\frac{i\pi kx}{N}} \tag{3}$$

Similarly in two dimensions we can write the continuous Fourier transform:

$$F(k,m) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{i2\pi kx} e^{i2\pi my} dxdy \tag{4}$$

And the inverse:

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(k,m) e^{-i2\pi kx} e^{-i2\pi my} dkdm \tag{5}$$

And we can discretise this as follows:

$$F(k,m) = \frac{1}{4NM} \sum_{x=-N}^{N-1} \sum_{y=-M}^{M-1} f(x,y) e^{i\pi kx} e^{i\pi my} \tag{6}$$

For the purposes of this exercise, I will only be using the real parts of the discrete Fourier transforms.

4 Problem A

4.1 Discrete Fourier Transforms of a finite slit

Here I create a single slit and its Fourier transform under various transformations of the slit.

To create the slit and Fourier transform, I used arrays. The indices range from 0 to 500, so I have transform from x to the index.

$$x = i - 250 \tag{7}$$

Where i is the array's index. In other words,

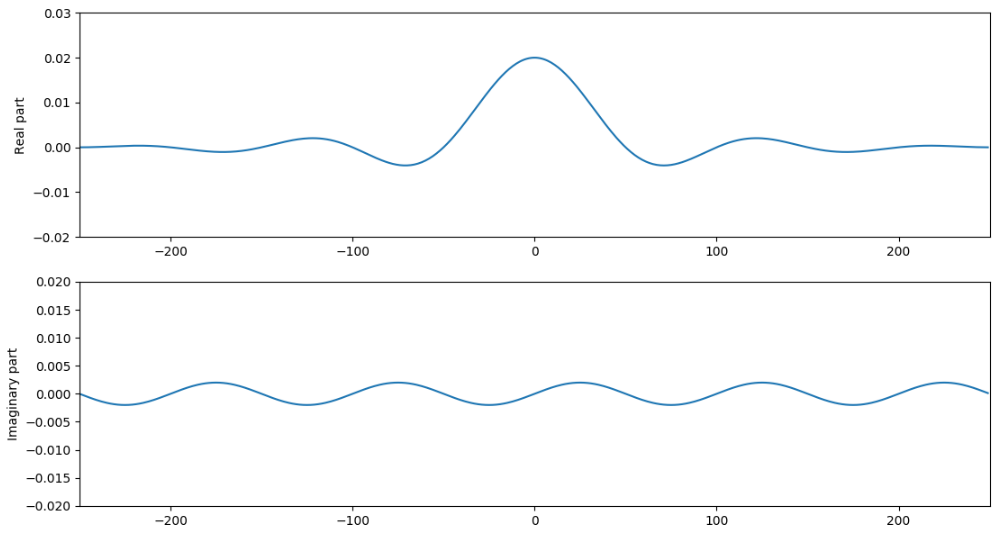
$$f(x) = a[x + 250] \tag{8}$$

Furthermore,

$$F(k) = b[k + 250] = b[j] \tag{9}$$

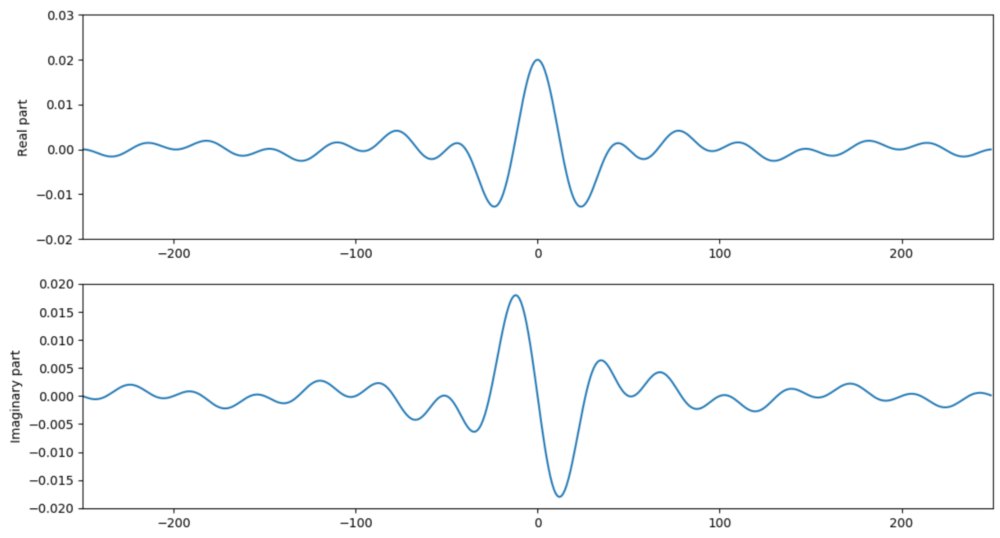
And to generate the DFT I iterate through j and sum over i.

The first task was to create an slit of width 10, height 1, centred on x = 0. I use an array here, making each element zero apart from those in the given range. This is the pattern that resulted:



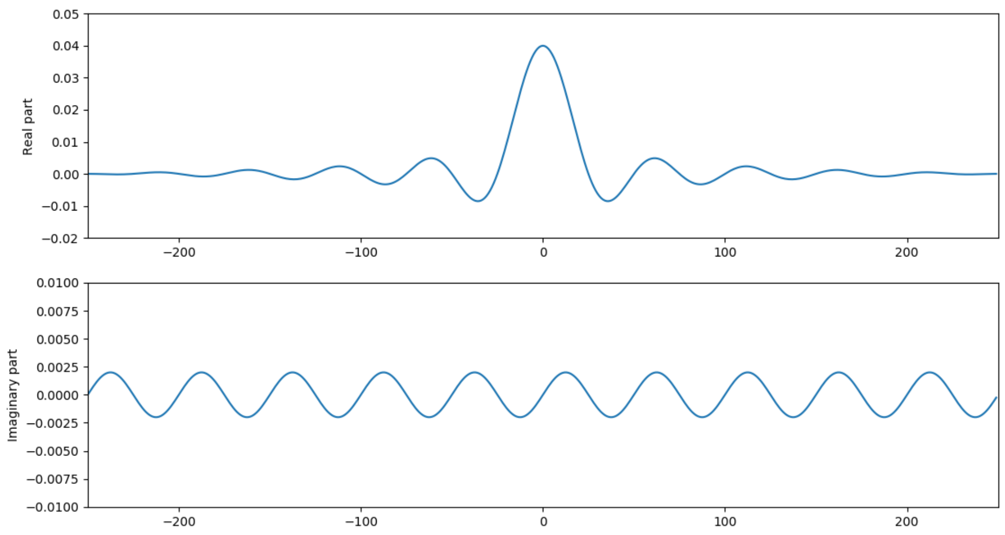
This is what I expected; it looks just like the Fourier transform plot for a single slit.

The second task was to shift this slit by 10 to the left. This is the Fourier transform of the array:



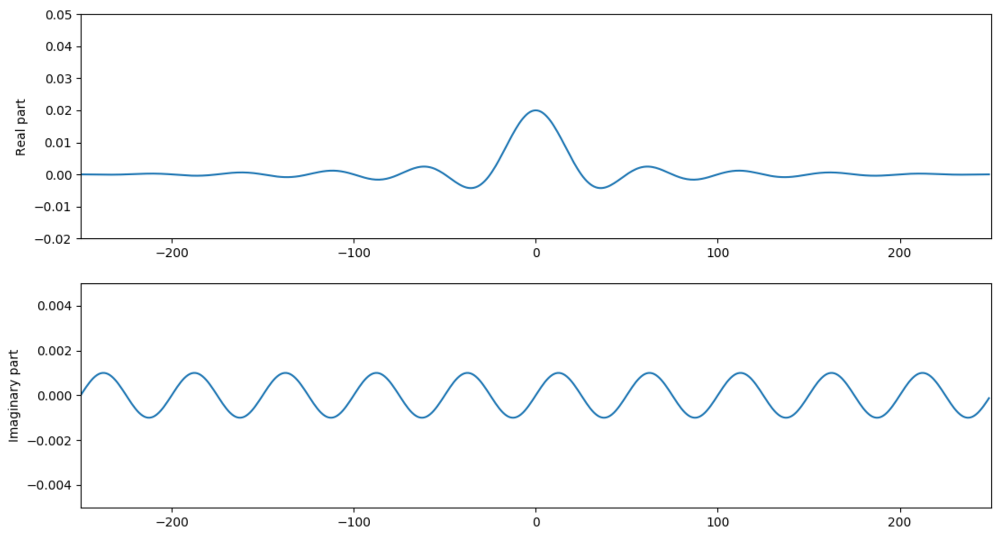
It is quite interesting that the frequency spread of the transform has altered so significantly, but it makes sense considering that the parity of the slit is no longer well-defined.

The third task was to double the width of the slit centred on the origin. This resulted:



This plot looks similar to the width-10 slit, but the frequency spread is narrower, again as expected.

Finally, I halved the height of this slit.



The amplitude has simply been scaled by 1/2, as the height of the slit does not affect the frequency distribution, just the amplitude of transmission.

4.2 The Convolution Theorem

The last part of this task is to verify the Convolution theorem.

The convolution of two functions $f(x)$ and $g(x)$ is given by

$$h(X) = \int_{-\infty}^{\infty} f(X)g(X-x)dx \tag{10}$$

The convolution theorem states that the Fourier transform of $h(x)$,

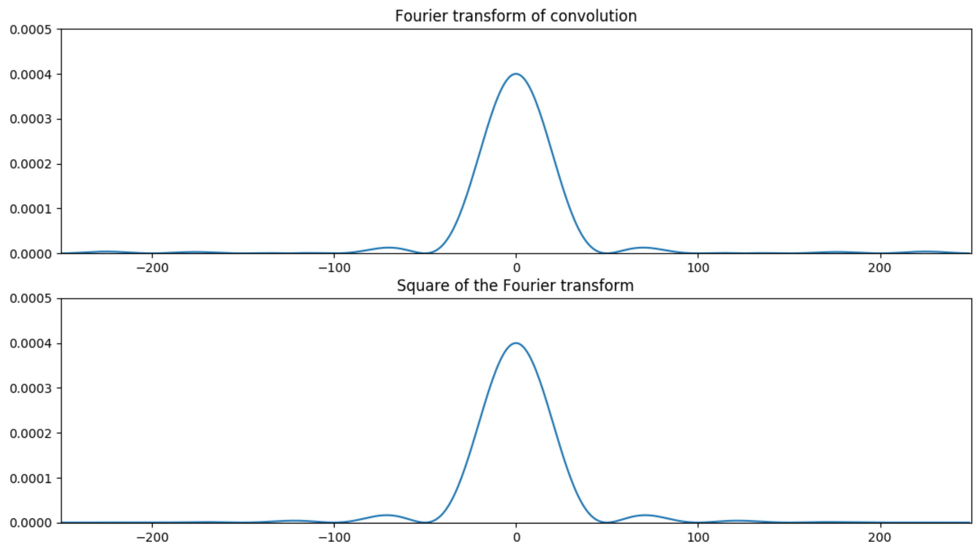
$$H(k) = F(k)G(k) \tag{11}$$

Where F and G are the Fourier transforms of f and g respectively.

In terms of the DFT,

$$H(k) = 2NF(k)G(k) \tag{12}$$

In this case, $f(x) = g(x) =$ the single slit. To compare, I have plotted both $F(k)^2$ and H(k) separately.



The plots are very similar; perhaps towards the extremities they deviate, but this can be accounted for by the discrete, approximate, nature of the convolution.

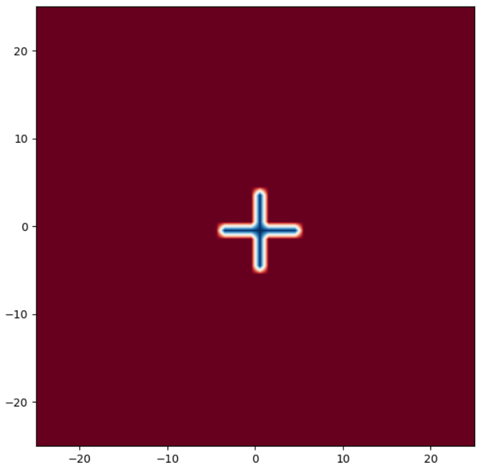
5 Optional Credit

In this section I will be executing 2D discrete Fourier transforms.

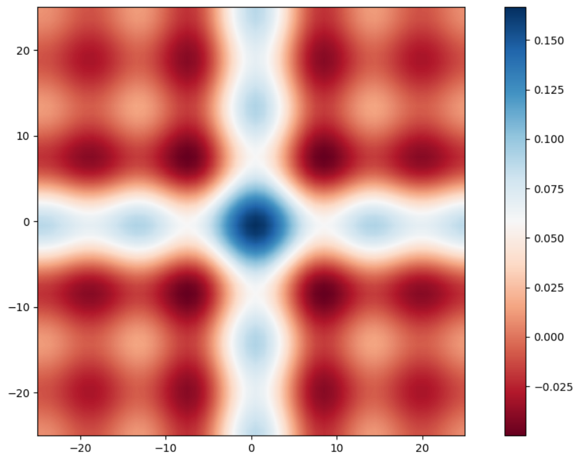
5.1 Problem A

For this problem I had to create a cross-shaped aperture and plot both the aperture and its 2D transform. The method was essentially the same as for the 1D plots, but iterating over two indices and two frequencies rather than just one of each.

This is the colour-plot of the aperture:



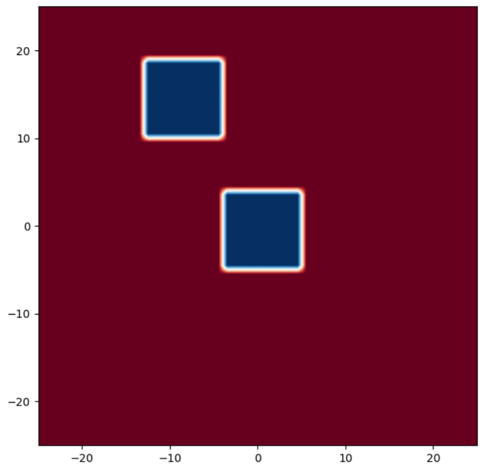
And this is the colour-plot of its Fourier transform:



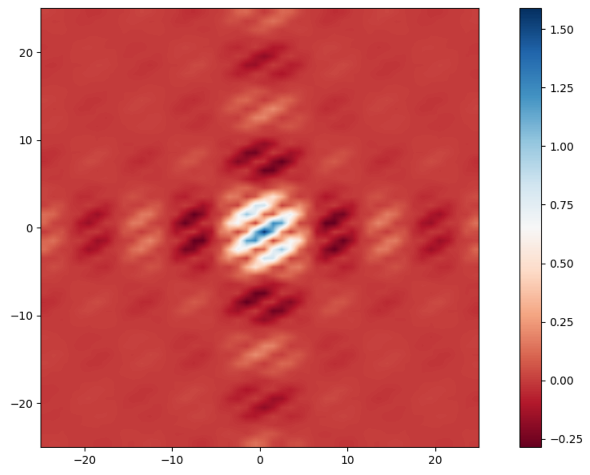
5.2 Problem B

For this problem I created two square sources.

This is a colour-plot of the aperture:



And this is its discrete Fourier transform:



I was not sure what to expect for the outcome of this transform, but it seems to be tilted in response to the relative position of the two square sources, which suggests that this is the correct result.

6 Summary

In conclusion, the discrete Fourier transform seems to be a powerful computational tool. However, I noticed that my programs were quite slow, especially for the 2D transforms, due to being very iteration-heavy. This was not helped by the fact that Python is an interpreted language; these tend to be slower compared to something like C because they are not optimised for speed. I wouldn't like to use my approach on a much larger aperture than what I have been using. If I tried to implement this recursively in something like LISP I would cause a stack overflow or several.

7 References

[1] E. Hecht - Optics 4th Edition - Addison Wesley

8 Appendix: Source code

Language: Python

8.1 Problem A

```
import matplotlib.pyplot as pl
import numpy as np
import math

def slmake(b, h, w):

    #Creates an array with values 0 representing opacity and higher values
    #some degree of transparency. The variable 'w' determines the width. 'b' determines
    #how left-centre the slit begins. 'h' determines the hight of the slit.

    slit = []
    for i in range(0, 500):
        slit.append(0)
        if i > 250 - b - math.ceil(w/2) and i <= 250 - b + math.ceil(w/2):
            slit[i] = h
    return slit

def dft(slit):

    #This function performs the discrete fourier transform on the slit created
    #above. RF is the real part; IF is the imaginary part.
    #The return value of this function is a 2D array, [RF, IF].

    RF = []
    IF = []

    for i in range(0, 500):
        RF.append(0)
        IF.append(0)

    for k in range(0, 500):
        #iterate through wavenumbers

        for j in range(0, 500):
            #iterate through x-values

            x = j - 250

            #these lines discretise the integral
            RF[k] += 1/500 * slit[j] * math.cos( (math.pi * x * (k-250))/250 )
            IF[k] += 1/500 * slit[j] * math.sin( (math.pi * x * (k-250))/250 )


    return (RF, IF)

def convolution(a, b):
    #simple algorithm to perform a discrete convolution of a and b

    con = []
    for _x in range(0, 500):
        con.append(0)
        for x in range(0, 500):
            con[_x] += a[x]*b[_x-x]

    return con

def main():
    r = []
    x = range(0, 500)
    for i in range(0,500):
        r.append(i - 250)


    #here I create a slit of height 1, left-shifted by 0, with width 10.
    F = dft(slmake(0, 1, 10))

    #these lines of code are just to make the plots of each part of the Fourier transform.
    #real part:
    pl.subplot(2, 1, 1)
    pl.plot(r, F[0])
    pl.ylabel("Real part")
    pl.axis([-250, 250, -0.02, 0.05])

    #imaginary part:
    pl. subplot(2, 1, 2)
    pl.plot(r, F[1])
    pl.ylabel("Imaginary part")
    pl.axis([-250, 250, -0.005, 0.005])
    pl.show()

    #this section of code is for the convolution exercise
    #I create a slit of height 1, left-shifted by 0, with width 10.
    slit = slmake(0, 1, 10)
    conslit = convolution(slit, slit)
    F2 = dft(conslit)
    #here I take the absolute value of the convolution's Fourier transform
    for m in range(0, 500):
        F2[0][m] = 1/500 * abs(F2[0][m])

    #this code is to plot the convolution's Fourier transform
    pl.subplot(2, 1, 1)
    pl.plot(r, F2[0])
    pl.title("Fourier transform of convolution")
    pl.axis([-250, 250, 0, 0.0005])

    #here I square the Fourier transform of the slit
    for k in range(0, 500):
        F[0][k] = F[0][k] * F[0][k]

    #now I plot it
    pl.subplot(2, 1, 2)
    pl.plot(r, F[0])
    pl.title("Square of the Fourier transform")
    pl.axis([-250, 250, 0, 0.0005])

    pl.show()


if __name__ == '__main__':
    #start the program
    main()
```

8.2 Optional Credit

```
import matplotlib.pyplot as pl
import numpy as np
import math

def makeslit(b1, b2, h, r):
    #b1: position of the centre of the first aperture
    #b2: position of the centre of the second aperture
    #h: height of each aperture
    #r: 'radius' of each aperture

    aperture_array = np.zeros((50, 50))

    #in the following code I simply iterate through and set all points within
    #the aperture to 1. In this case I create two square sources.

    for i in range(0, 50):

        for j in range(0, 50):

            if i == 25 - b1[0] and j == 25 - b1[1]:

                for k in range(0, r):
                    for l in range(0, r):
                        aperture_array[i-k][j-l] = 1
                        aperture_array[i+k][j-l] = 1
                        aperture_array[i-k][j+l] = 1
                        aperture_array[i+k][j+l] = 1

            elif i == 25 - b2[0] and j == 25 - b2[1]:
                for k in range(0, r):
                    for l in range(0, r):
                        aperture_array[i-k][j-l] = 1
                        aperture_array[i+k][j-l] = 1
                        aperture_array[i-k][j+l] = 1
                        aperture_array[i+k][j+l] = 1


    return aperture_array

def dft(arr):

    F = np.zeros((50,50))

    #this is a simple 2D, real Fourier transform of a 2D array, 'arr'.
    #the algorithm is the one I covered in the introduction of this report.

    for m in range(0, 50):
        k = m - 25
        for n in range(0, 50):
            l = n - 25
            for i in range(0, 50):
                for j in range(0, 50):
                    x = i - 25
                    y = j - 25

                    F[n][m] += 1/(4*25^2) * arr[j][i] * math.cos( (math.pi*(x*k + y*l))/25)

    #set the Fourier transform to its absolute values
    for M in range(0, 50):
        for N in range(0, 50):
            F[n][m] = abs(F[n][m])

    return(F)

def main():
    #here all the busywork is done: mostly, plotting.
    z = np.zeros((50,50))
    d = np.zeros((50,50))
    for i in range(0, 50):
        for j in range(0, 50):
            z[i][j] = i + j - 50
            d[i][j] = i + j - 50

    Sl = makeslit([0, 0], [15, 9], 1, 5)

    F = dft(Sl)

    #here I plot F with a red-blue colour map to represent its amplitude.

    fig, ax = pl.subplots()
    t = pl.imshow(F, cmap=pl.cm.RdBu, vmin=F.min(), vmax=F.max(), extent=[-25,25,-25,25])
    t.set_interpolation('bilinear')
    c = fig.colorbar(t)
    pl.show()

    #here I plot the aperture, also with a red-blue colour map.

    tt = pl.imshow(Sl, cmap=pl.cm.RdBu, vmin = Sl.min(), vmax=Sl.max(), extent=[-25, 25, -25, 25])
    tt.set_interpolation('bilinear')
    cc = fig.colorbar(tt)
    pl.show()

if __name__ == '__main__':
    #start the program.
    main()
```