CS 474/674 - Image Processing and Interpretation

Programming Assignment 3

Submitted by-

Andrew Wiltberger and Akshay Krishna

Due Date: November 18th, 2020

Handover Date: November 16th, 2020

Division of Work:

Andrew Wiltberger and Akshay Krishna split the programming and report writing equally. Andrew covered Experiment 1 and Experiment 3A. Simultaneously, Akshay covered Experiment 2 and Experiment 3B. Both Andrew and Akshay contributed to the sections: Theory, Implementation, Results, and Discussion of their respective topics.

Theory:

Experiment 1:

The first experiment is about the 1-dimensional Fourier Transform (FT). This transform and the Inverse Fourier Transform (IFT) are used in image processing to convert from the spital to the frequency domain. This conversion is useful because many operations, such as convolutions, can be computed more quickly in the frequency domain. However, to have real-world applications, the discrete versions of the FT and IFT are used. Both are defined below by eq. 1 and eq. 2, respectively.

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)e^{\frac{-j2\pi x}{N}},\ \ u = 0,1,2,\dots,N-1\ \ (eq.1)$$

.

$$f(x) = \sum_{u=0}^{N-1} F(u)e^{\frac{j2\pi x}{N}},\ \ u = 0,1,2,\dots,N-1\,(eq.2)$$

The discrete version of the FT (DFT) and IFT (DIFT) can convert an image or any signal from the spital domain to the frequency domain. However, in this form, the DFT and DIFT have a run time of $O(N^2)$, which is expensive. The Fast Fourier Transform (FFT) is a divide and conquer algorithm that can compute the transform in $O(NlogN)$. The FFT makes the assumptions the number of samples it will be operating on is a power of 2 and splits the summation for the DFT into two parts, one for even terms one for odd terms as shown in eq. 3 and eq. 4.

$$F(u) = \frac{1}{2M}\sum_{x=0}^{2M-1} f(x)W_{2M}^{ux},\ \ W_M = e^{\frac{-2j\pi}{N}}\ \ 2M = N\ \ (eq.3)$$

$$F(u) = \frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1} f(2x)W_{2M}^{u(2x)} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)W_{2M}^{u(2x+1)}\right](eq.4)$$

By expanding the powers of W and simplifying, we get eq. 5 where F_{even} is the even terms and F_{odd} is the odd terms.

$$F(u) = \frac{1}{2}[F_{even}(u) + F_{odd}(u)W_{2M}^{u}], u = 0,1,\dots,M-1\,(eq.5)$$

Eq. 5 will only give the DFT for the first half of elements *0,1…,M-1*. To find the other half of the transform, *M,M+1,…,2M-1*, we must compute *F(u+M)*, shown in eq. 6.

$$F(u+M) = \frac{1}{2}[F_{even}(u+M) + F_{odd}(u+M)W_{2M}^{u+M}],(eq.6)$$

This equation is then simplified to eq.7.

$$F(u) = \frac{1}{2}[F_{even}(u) - F_{odd}(u)W_{2M}^{u}], u = M, M+1,\dots,2M-1\,(eq.7)$$

From eq. 5 and eq. 7, It is clear to see we can compute the DFT of a length N sample, where N=2M, by finding the DFT of the odd and even terms for the first half of the samples and then changing the sign of the odd terms of the second half. The IDFT can be computed in a similar way but with a sign change to the complex exponential. In the first experiment, an FFT routine was

used to compute the DFT of the small sample, samples from cos(), and samples from the rectangle function. The results of these experiments will be discussed later in this report.

Experiment 2:

The focus of experiment 2 is on 2-D DFT. The 2-D DFT can be computed by extending the 1-D DFT, which is represented in eq. 8.

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{x=0}^{N-1} f(x,y)e^{\frac{-j2\pi(ux+vy)}{N}},\ u,v = 0,1,2,\dots,N-1\ (eq.8)$$

The equation shown above has a complexity of $O(N^4)$, assuming the image to be a square image of dimensions N x N. To compute this faster, we can take advantage of the DFT properties of separability to rewrite eq.8 as eq. 9

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1} e^{-j2\pi\frac{ux}{N}} \sum_{x=0}^{N-1} f(x,y)e^{-j2\pi\frac{vy}{N}}, (eq.9)$$

We can see that the right summation is N times the DFT of the rows, and when we represent the right summation as $F(x, v)$, we get eq. 10.

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1} e^{-j2\pi\frac{ux}{N}}\ F(x,v)\ (eq.10)$$

From eq. 10, it is clear to see that to compute $F(u, v)$, you must first compute $F(x, v)$ by finding N x DFT of the rows of $f(x, y)$. Then, we have to finally apply the DFT to the columns of $F(x, v)$. This procedure will have a complexity of $O(N^3)$.

When we compute $F(u, v)$, we cannot see the full period. To see the full period, we need to translate the origin of the signal $F(u, v)$ by $(N/2, N/2)$. This translation is achieved by following the frequency domain shifting, which is represented in eq. 11. To move $F(u, v)$ at $(N/2, N/2)$, we take $u_0 = v_0 = N/2$.

$$f(x,y)e^{\frac{j2\pi(u_0x+\ v_0y)}{N}}" = " F(u-\ u_0, v-\ v_0),\ \text{(eq. 11)}$$

When we substitute $N/2$ in u_0 and v_0 of the exponential term, we get a real number, represented in eq. 12.

$$e^{j2\pi(\frac{\frac{N}{2}x+\frac{N}{2}y}{N})} =\ e^{j\pi(x+y)} = (-1)^{x+y},\ \text{(eq. 12)}$$

Therefore, to shift the signal in the frequency domain $F(u, v)$ to the coordinates $(N/2, N/2)$, we multiply the input signal with $(-1)^{x+y}$, where x and y are the spatial locations. This is represented mathematically in eq. 13.

$$f(x,y)(-1)^{x+y}" = " F\left(u-\frac{N}{2}, v-\frac{N}{2}\right),\ \text{(eq. 13)}$$

Experiment 3:

The focus of experiment 3 is to determine the importance of the phase and magnitude information from the DFT of *f(x, y)*. From the property of symmetry, we know that if the input to the DFT *f(x, y)* is real, then the transformed 2D signal *F(u, v)* is a complex signal, which can be represented as *R(u, v) + jI(u, v)*. Here, the real part *R(u, v)* is an even signal, and the imaginary part *I(u, v)* is an odd signal. This transformation is represented in eq. 14. In our experiment, we consider the 2D input to be an image, and we know that images are real-valued signals.

$$f(x,y)real \ \ll\!=\!\gg R(u,v)even + jI(u,v)odd, \text{ (eq. 14)}$$

We can calculate the magnitude and the phase of the frequency domain signal *F(u, v)*, using the real *R(u, v)* and the imaginary *I(u, v)* part of the signal. The magnitude can be calculated using eq. 15., where *Mag[F(u, v)]* represents the magnitude of *F(u, v)*.

$$Mag[F(u,v)] = \sqrt{R(u,v)^2 + I(u,v)^2}, \text{ (eq. 15)}$$

Similarly, the phase of *F(u, v)*, which is represented as *θ[F(u, v)]*, can be calculated using the mathematical formula in eq. 16.

$$\theta[F(u,v)] = \tan^{-1}\left(\frac{I(u,v)}{R(u,v)}\right), \text{ (eq. 16)}$$

To determine if the magnitude information is more important than the phase information, we perform DIFT on the modified version of the DFT of *f(x, y)*. To retain the magnitude information, we set the phase of the signal to zero. This operation is done by setting *I(u, v)* to zero and *R(u, v)* to *Mag[F(u, v)]*. By performing these changes and substituting in eq. 15 and eq 16., we get the following results represented in eq. 17 and eq. 18.

$$\sqrt{Mag[F(u,v)]^2 + 0} = Mag[F(u,v)], \text{ (eq. 17)}$$

$$\tan^{-1}\left(\frac{0}{Mag[F(u,v)]}\right) = 0, \text{ (eq. 18)}$$

From the above equations, eq. 17 and eq. 18, it is evident that the magnitude is retained, and the phase is set to zero. Now, to retain the phase information and to discard the magnitude information we set *R(u, v) = cos(θ[F(u, v)])* and *I(u, v) = sin(θ[F(u, v)])* and substitute these changes in eq. 15 and eq. 16.

$$Mag[F(u,v)] = \sqrt{\cos(\theta[F(u,v)])^2 + \sin(\theta[F(u,v)])^2} = 1, \text{ (eq. 19)}$$

$$\tan^{-1}\left(\frac{\sin(\theta[F(u,v)])}{\cos(\theta[F(u,v)])}\right) = \theta[F(u,v)], \text{ (eq. 20)}$$

From the above equations, eq. 19 and eq. 20, it is evident that the phase information is retained, and the magnitude is set to 1 throughout the image. We reconstruct the image with these changes and determine which is more important.

Implementation:

Experiment 1A.

Functionality for part A is achieved with the provided *fft( )* routine, simple if/else, and for loops. For part a, the signal was hardcoded into a float array as it was only 8 elements long. The *fft* routine was then called on the array. To print the results of the *fft,* a for loop was used to iterate through the relevant index of the array *(data[1] – data[2nn]).* In the for loop, we multiply every element in the array by .25 to normalize the elements. If the element is odd, we will print the real part to the file. We will also compute the magnitude and phase parts and print them to their respective files. If the index is even, then we will print the imaginary part to a file. After that, we compute the inverse *fft* and print the results to files with the same procedure as the forward *fft*.

Experiment 1B.

Functionality for part B is achieved with the provided *fft( )* routine, simple if/else, and for loops. For part B, the samples will be held in a float vector. To fill the vector with a sample of cos, we will call the *sampleCos( )* function. This function will run a for loop and will iterate 128 times. At even indices of *i*, a sample of *cos(2*M_PI*8*i/128)* will be placed, and at odd indices, 0 will be placed. After the for loop is done iterating, it will return the vector with the proper samples.

Next, we will flip every other real part to shift the result for visualization purposes and call the *fft( )* routine. However, because *[0]* holds a value we care about we call the function as follows: *fft(&samples[0]-1, samples.size( ), -1).* After we get a result, we will normalize every element by dividing by *N*. After that, and we apply thresholding to the samples. This thresholding is done because there are many round-off errors when using floats, and we may have a minimum value when a number should be 0. The value of the threshold is $1.0e^{-6}$. Finally, the real, imaginary, phase and magnitude parts are printed to files.

Experiment 1C.

Functionality for part C is achieved with the provided *fft( )* routine, simple if/else, and for loops. To create a rectangle function sample, we used the *getline( )* function to read the file containing the rectangle function. We placed the samples from the file into even indices and 0 into odd indices in a float vector. We will then flip every real value and similarly call the fft routine to Experiment 1, part B. After that, we will normalize every element and print the outputs into their respective files.

Experiment 2:

All the parts in this experiment used the same implementation but with different sized squares placed at the center of the image. We first prompt the user to input the white square's size and a constant for contrast stretching. Next, we calculate the start and stop index using the size of the square. After that, we iterate through the image using nested for loops, and if the index falls within the square, we will set the pixel value to *255*pow(-1, i+j)* for the shifted version and *255* for the non-shifted version. If the index is not within the square's bounds, we will set the value to 0 in the normal and shifted images.

After that we call the *fft2D( )* function. The *fft2D( )* function first calculates the fft of the rows with the *rowfft* function. This function copies the real and imaginary parts into one array and then calls the provided *fft( )* routines, fills the real and imaginary arrays with the results, and returns to the *fft2D( )*. We now have *F(x, v)*. To compute *F(u, v),* we call the *colfft* function. This function does the same thing as *rowfft* but concerning the columns. We then return to the main results of the 2D fft.

We then compute the magnitude part by iterating through every element of the transformed image by first computing *std::pow(std::pow(int(real[i][j]), 2) + std::pow(int(imag[i][j]), 2), 0.5))* and then applying *Clog2(1+mag[i][j])* to accomplish contrast stretching. After that we write the transformed image.

Finally, we apply the *fft2D( )* and compute the shifted image's magnitude and write the results.

Experiment 3:

First, we check if the image is a square image and whether the image's dimensions are a power of 2 by calling the *check_dimensions( )*. If the image's dimension is not a power of 2, then we return a greater number that is closest and is a power of 2 of the dimensions to perform zero paddings. We calculate the start and stop index to stop padding zeros at these locations and copy the actual image. We follow the same steps as we did in Experiment 2 to compute the DFT of the image without performing any sort of translation or shifting.

Once the 2D DFT is computed, we set the transformed image phase to zero by calling the *setMagnitude( )* function. Once the phase is set to zero, we compute the DIFT, trying to reconstruct back the original image. The phase is set to zero by following the equations mentioned in the theory section. We follow the same steps to discard the magnitude information but by calling the *setPhase( )* function. This function discards the magnitude information from the DFT signal by following the theory section's equations. Once the magnitude information is discarded, we try to reconstruct the original signal by calling the IDFT.

We perform IDFT by calling the specified *fft( )* routine and initializing the *isign* argument to 1. For visualization, we compute the magnitude of the IDFT signal by calling the *magnitude( )* function. If we are discarding the magnitude, we call the *mapper( )* function after calling the *magnitude( )*. This function is used to bring the pixel range in the image between [0, 255]. Then we finally create the images using the *writeImage( )* function.

Results:

Experiment 1A.

The results of part A are shown in Fig. 1. As expected, when the DFT and IDFT are applied, we get the original sample back. When first computing this when applying the IDFT, the answer would still have small values for the imaginary part when they should have been zero. This is due to the computer's floating-point error and can be solved by setting a threshold to eliminate these errors.

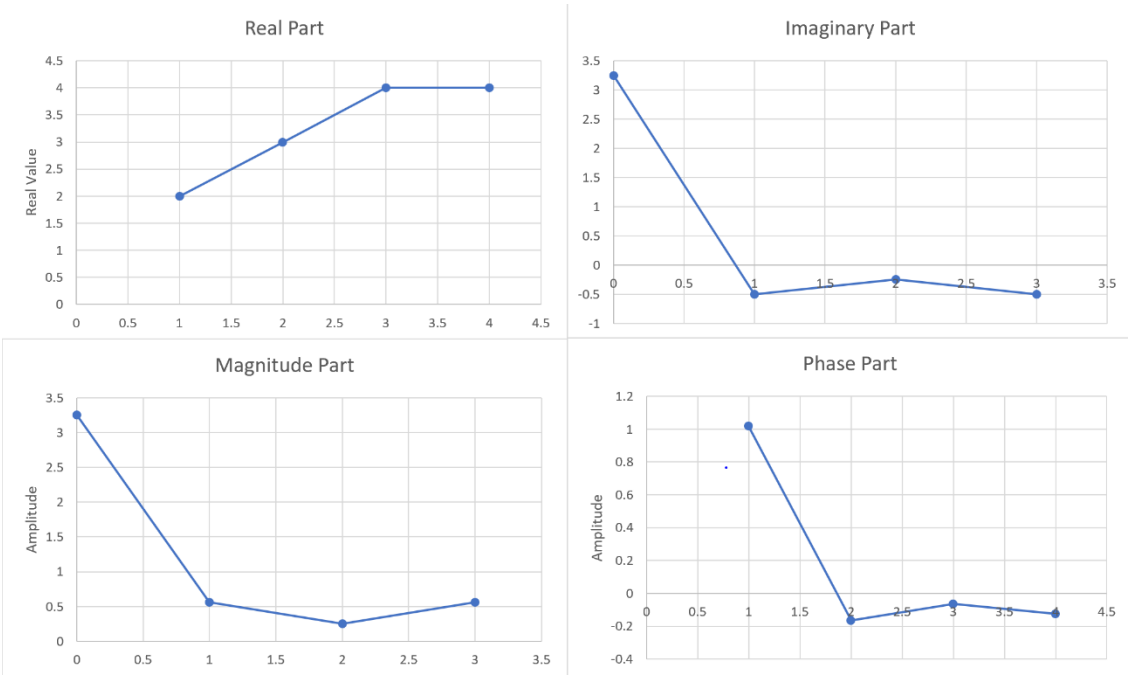

Fig.1 Plots of all parts of DFT.

Experiment 1B.

The samples that were used for this experiment are shown below in Fig. 2. Clearly, Fig 2. is the correct sample as there are 8 periods of the cosine function. This produced the expected results where the spikes are at -8 and 8 in the real plot. However, the imaginary plot is always 0, as shown in Fig. 3. The phase and magnitude provide no other information due to the imaginary part being 0. They can be referenced in Fig 4.

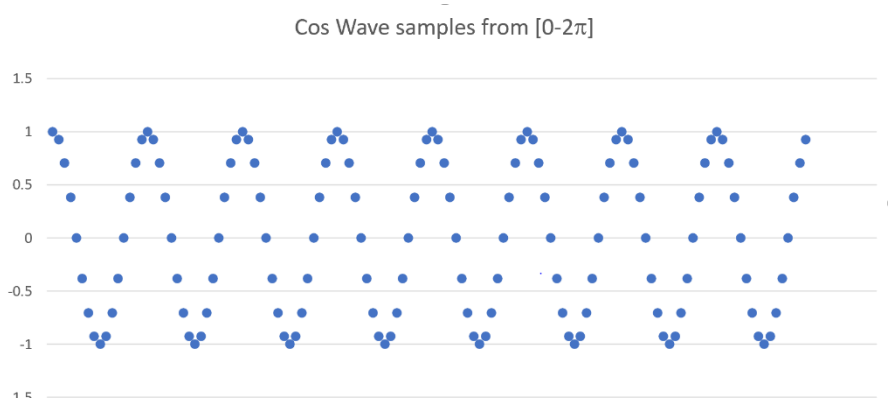

Fig. 2 Samples of cos

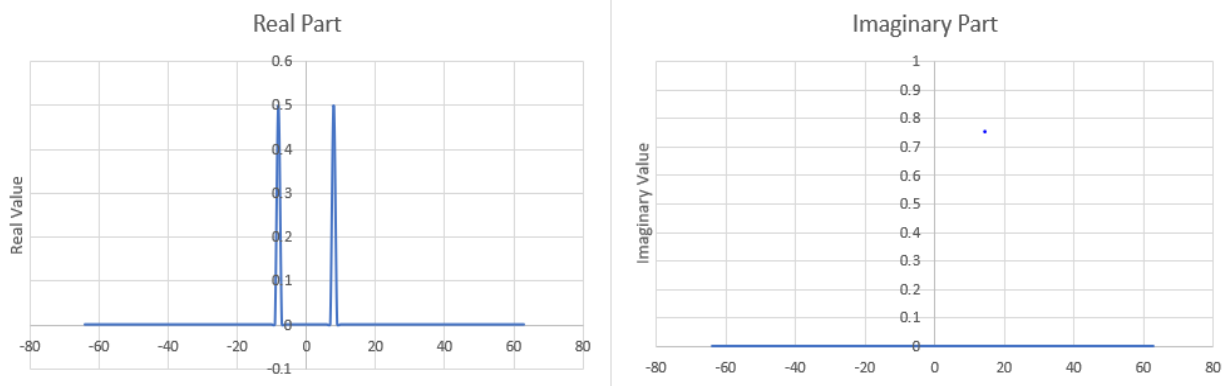

Fig. 3 Plots of real and imargary parts of DFT

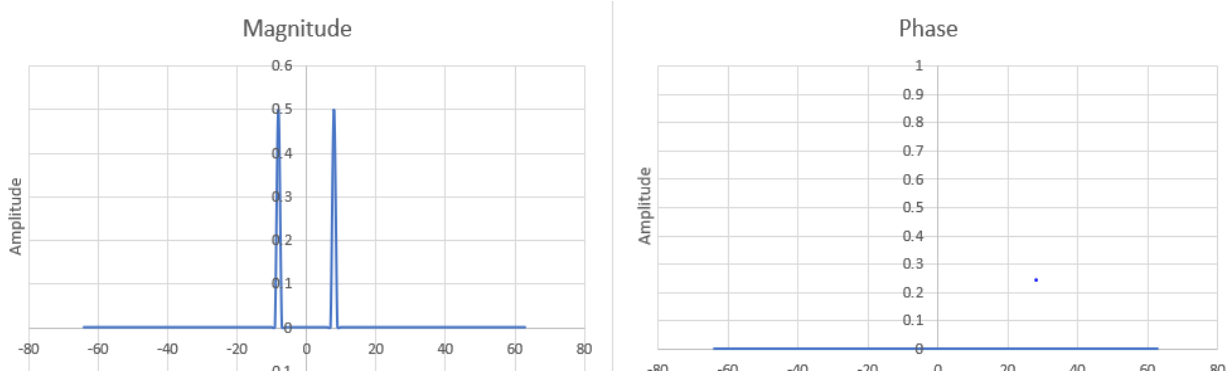

Fig. 4 Plots of magnitude and phase of DFT

Experiment 1C.

In part C of Experiment 1, we applied the DFT to the rectangular function Fig. 5. The expected result is a sinc function. The real part Fig. 6 matches with the sinc function. However, the imaginary part is not 0. This is due to floating-point errors in the program. This is supported by the fact that the imaginary parts are so small. The magnitude matches the expected result of the absolute value of the sinc function, and the errors in the phase components are due to the floating-point errors.

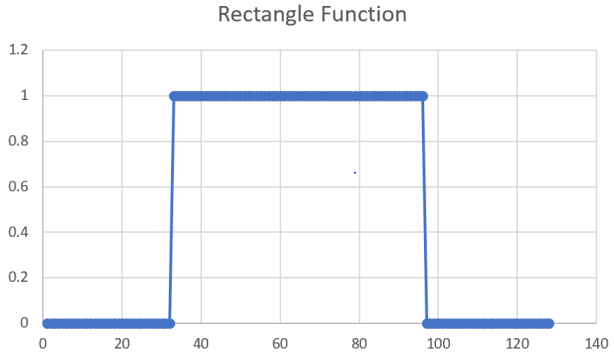

Fig. 5 Original Rectangle Function

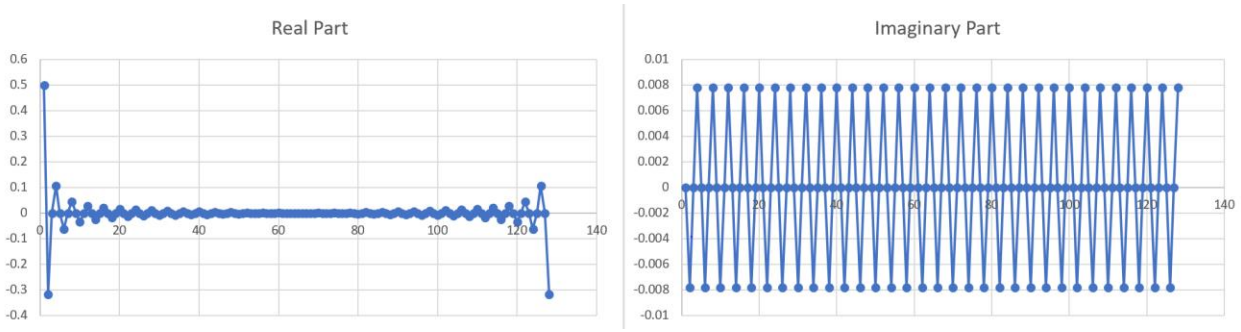

Fig. 6 plots of real and imaginary part of DFT

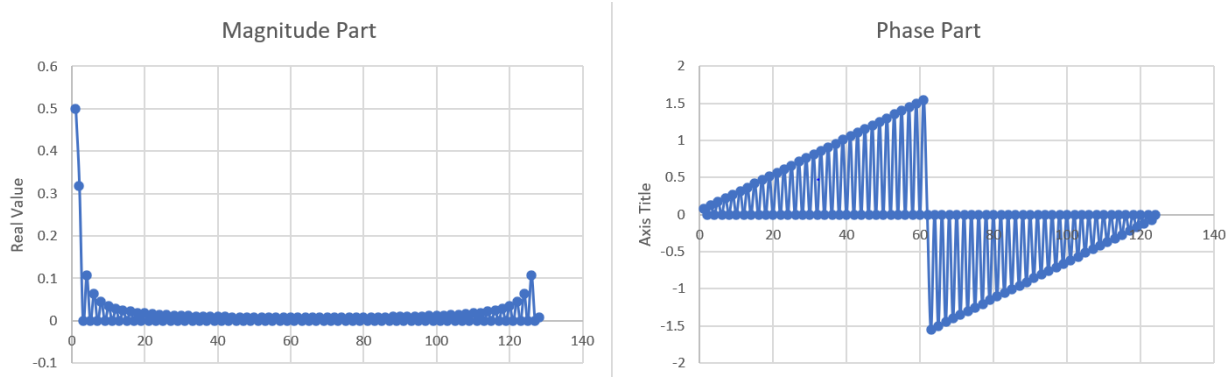

Fig. 7 plots of the magnitude and phase parts of DFT

Experiment 2

In Experiment 2, we performed the 2D DFT on black images with varying white squares in the center. Once the magnitude has been shifted, the DFT output is a white cross across the center of the image. The intensity of the white pixels increases as they get closer to the middle of the image. We know that the DFT of the rectangle function is the sinc function. The square in the center can be considered as the 2D rectangle function and the resulting DFT as the 2D sinc. From the shifted 2D DFT image, it is quite evident that the original image consists of multiple varying frequencies, but mainly consisting of lower frequencies. We came to this conclusion because the signal to the center is bright (corresponding to low frequencies), and the brightness decreases as we move away from the center (corresponding to high frequencies). As the size of the square increases in the original image, the number of zero crossings increases in the 2D DFT image, which can be seen in the results. The results of this experiment can be found in Fig. 8, Fig. 9, and Fig. 10. The low frequencies in the image correspond to a gradual change of intensities in the image and high frequencies correspond to rapid change in intensities in the image like edges and noise. As there are not many rapid changes in the original image, the image mainly consists of low-frequency components.

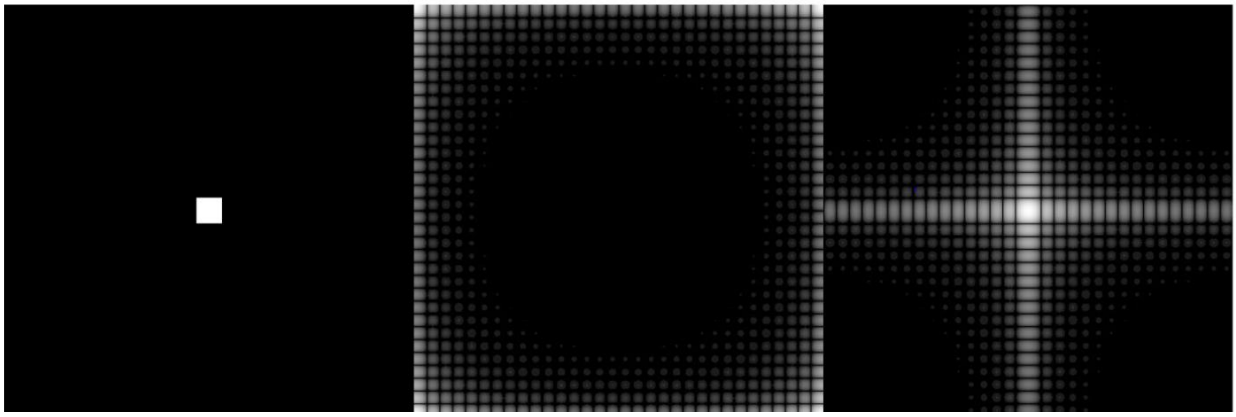

Fig. 8 From left to right original image, magnitude, and shifted magnitude of 32x32 white square

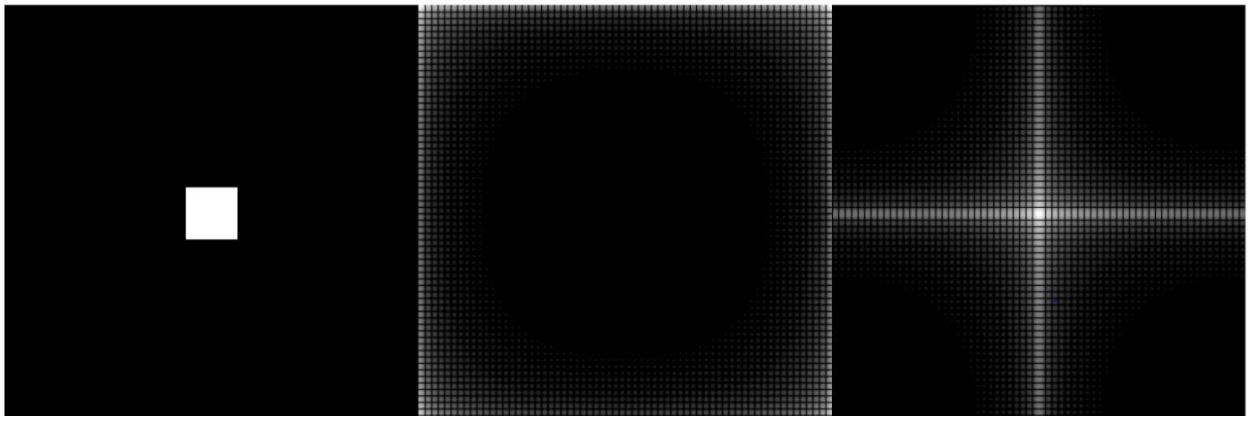

Fig. 9 From left to right original image, magnitude, and shifted magnitude of 64x64 white square

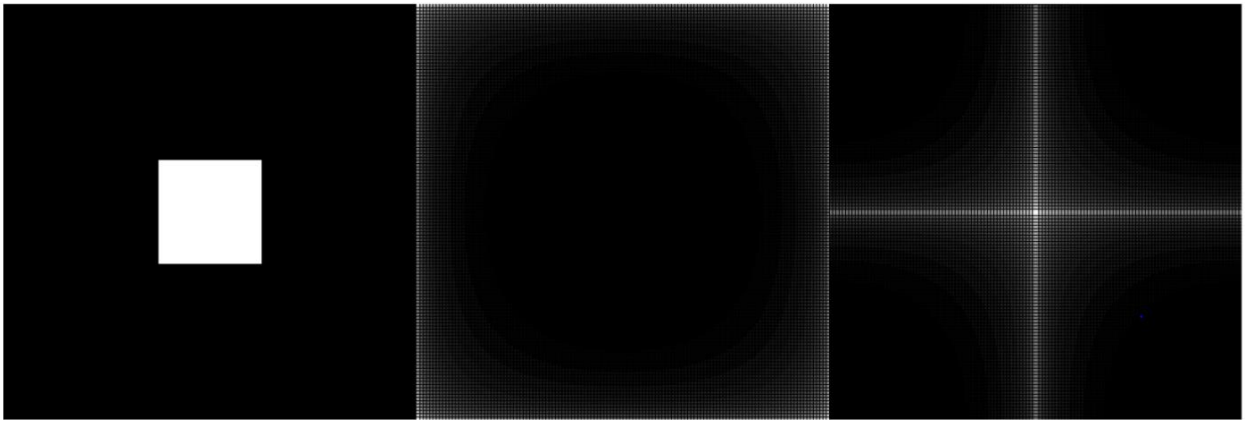

Fig. 10 From left to right original image, magnitude, and shifted magnitude of 128x128 white square

Experiment 3:

Experiment 3 determines the importance of the magnitude and phase information present in the 2D DFT. From the results, it is quite evident that the phase information is more important than the magnitude information. We came to this conclusion because, when we tried to reconstruct the image when we discarded the magnitude, we can still see the structure of the original image being maintained in the reconstructed image. But when we discard the phase information from the DFT and reconstruct the image back, no sense is being made by the image. The results of this experiment can be seen in Fig. 11.

Fig. 11 From left to right original image, magnitude discarded image, and phase discarded image