Computing the 2-D Discrete Fourier Transform

I. INTRODUCTION

The Fourier transform was initially based on the concepts of Jean Baptiste Joseph Fourier who in 1822 published the paper "*Théorie analytique de la chaleur*". Within this paper he develops the idea that all discontinuous functions could be expressed as a series of sinusoidal waves. His formulation was flawed; however, this idea was a breakthrough and led to the development of the Fourier series. The Fourier transform of expresses a signal as a composition of sinusoidal functions.

With the development of the computer during the mid 20th century there came a need for a quick method of determining the discrete Fourier transform of a signal. Initially these methods were very processor and memory intense. With the computer technology of the mid 20th century, this meant very large computer and lots of time (24 hrs for a 128 x 128 image!). To decrease the time it took to process these discrete transforms many versions of the Fast Fourier Transform (FFT) were created. The mostly widely used algorithm was developed in 1965 by J.W. Cooley and John Tukey name the Cooley-Tukey algorithm. This algorithm reduced the FFT computation time to O(nlog(n)) from a $O(n^2)$. The version of the FFT implemented in MATLAB is largely based on the Cooley-Tukey algorithm with other optimizations. The execution time for the fft depends on the length of the transform. It is fastest for powers of two. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or which have large prime factors.

II. IMPLEMENTATION OF THE FFT2 IN MATLAB

The MATLAB software environment provides a powerful set of tools for performing signal analysis and other applications. This tool set comes with a learning curve that can lead to some non-intuitive results when one first attempts an operation. This is very much the case for the implementation of the fft2 MATLAB function. This following discussion provides a guideline of the basic operation of the function and shows howone can sweet-talk the function into producing the outputs we desire.

The first of these problems arises with the special layout of the input and the output of these functions. Many of us have become familiar with the frequency space results of the one

dimensional Fourier transform. We expect that from most signals there will be a relatively large 'DC' component located at the origin and the transform will be mirrored about the origin. This holds true for the two dimensional case where we find the 'DC' component located at the origin of the two dimensional plane and the frequency space function is mirrored about this point. This however is not the result of directly using the fft2 function without any pre or post processing.

First, we need to create a simple function to be transformed. For this example, a simple 2-D rect function will be used. In order to place our rect in exactly in the center of our matrix we need to define both the rect and the matrix with odd dimensions. We can define our rect using the following MATLAB commands.

```
rect = zeros(51);
rect(25:35,25:35)=ones(11);
surf(rect)
```

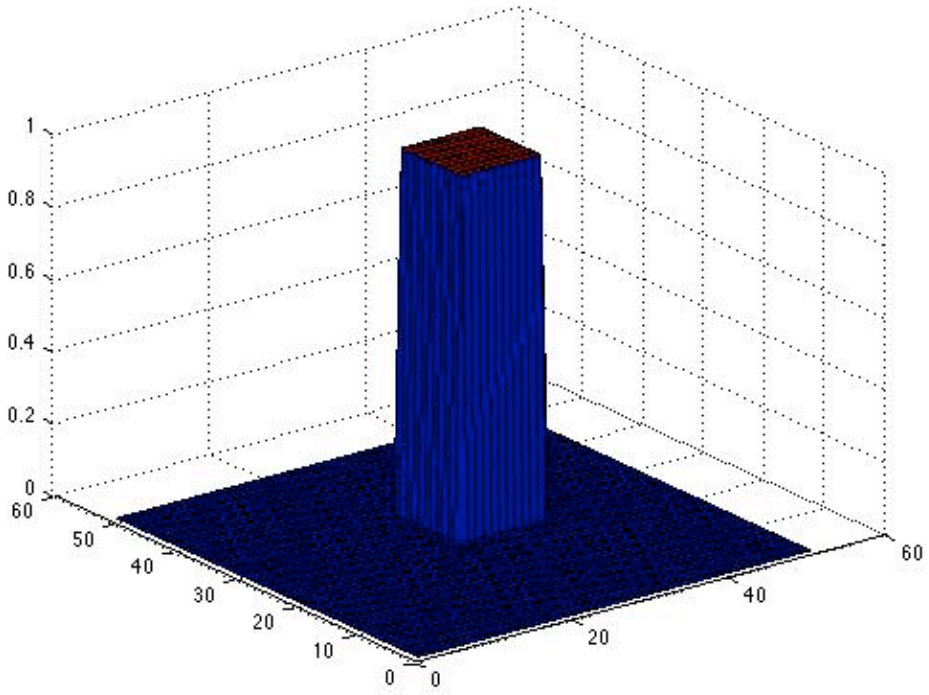

Figure 1: The mesh of a two dimensional rect function 11x11 units wide. This function uses an odd number for its side and its image to ensure that it is exactly centered.

Now the fft2 command can applied to this rect.

```
rect_fft = fft2(rect);
```

These complex functions are commonly viewed as magnitude plots. To view the frequency space function the following command should be used.

```
surf(abs(rect_fft))
```

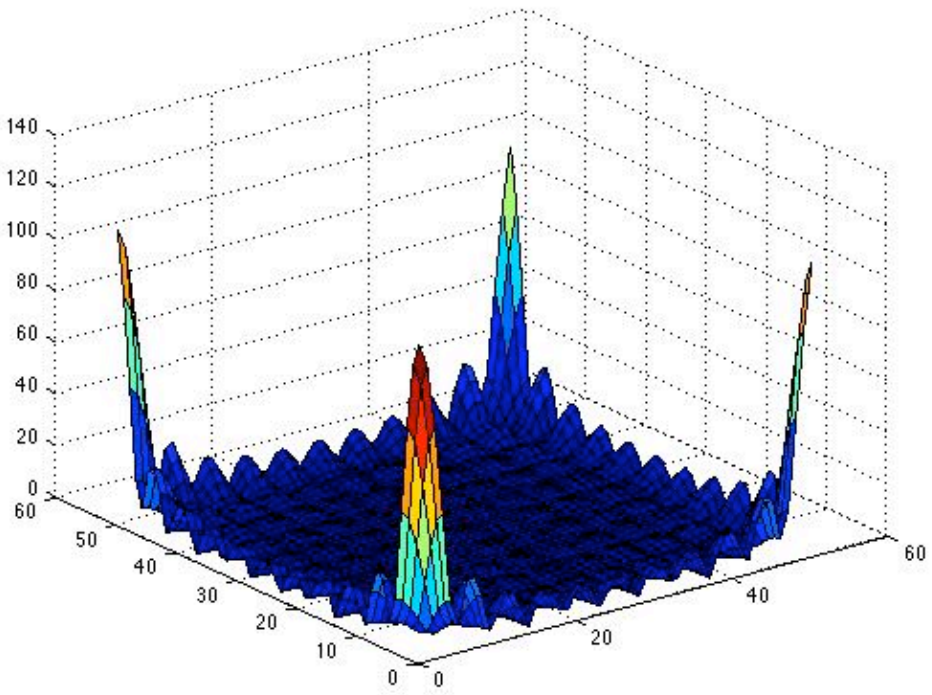

Figure 2: The direct fft2 of the rect function in Figure 1. The DC components of the frequency space plot have been placed in the corners of the matrix.

This result is not what has come to be expected from experience with Fourier Transforms. We expect that the low frequency components should be located near the center of the output matrix and not the edges. MATLAB supplies functions that both pre and post process the image both in image space and frequency space so that the expected results are obtained. The MATLAB function fftshift can be used to shift both the input and the output of the fft2 function. The following discussion shows how this can be done for the previously defined rect function.

First the rect will be shifted in image space.

```
rectshifted=fftshift(rect);
surf(abs(rect_shifted))
```

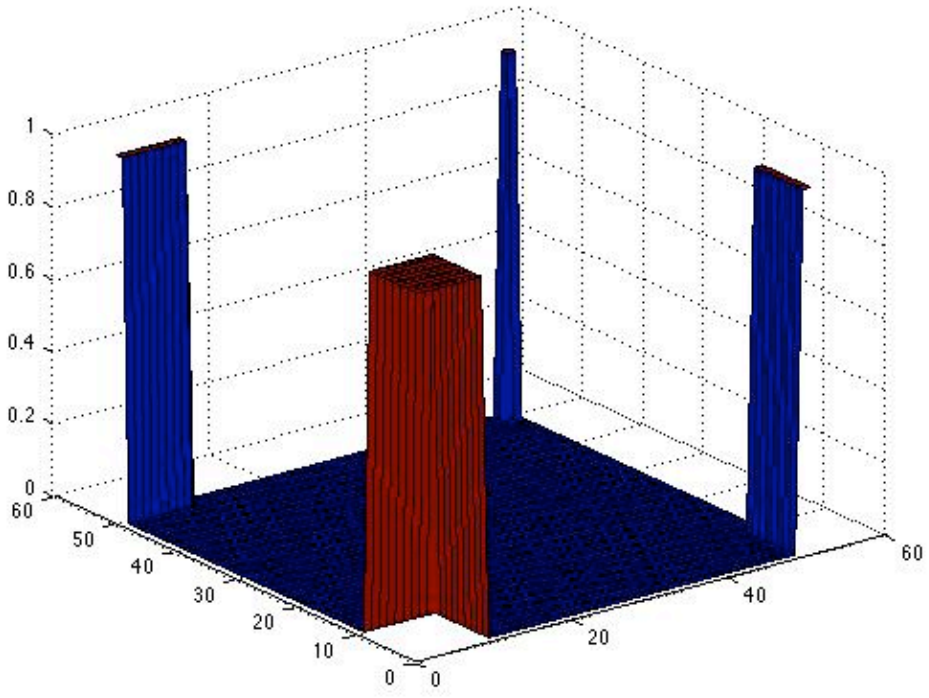

Figure 3: The rect function after it has been shifted using the fftshift command. If this image was replicated in both the x and y axis, it would still remain continuous at the boundaries.

Now that we have shifted the image, we apply the two dimensional Fourier Transform to it.

```
rectfft=fft2(rectshifted);
```

Now after a post processing shift, again using the fftshift command, the expected 2-D sinc pattern can be seen.

```
shiftedrectfft=fftshift(rectfft);
surf(abs(shiftedrectfft))
```

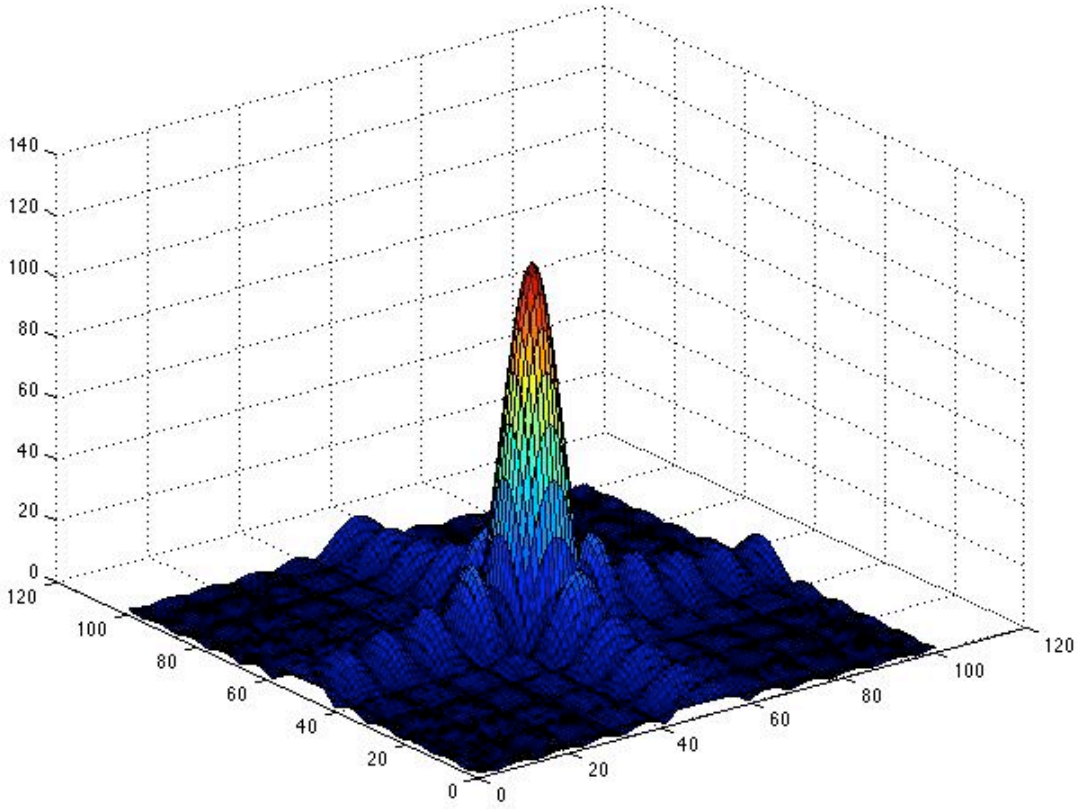

Figure 4: After applying the fftshift command to the Fourier Transform, the image looks a bit more familiar.

Full code:

```
rect=zeros(101);
rect(45:55,45:55)=ones(11);
rectshifted=fftshift(rect);
rectfft=fft2(rectshifted);
shiftedrectfft=fftshift(rectfft);
figure
surf(abs(shiftedrectfft))
colormap(hsv)
figure
pcolor(abs(shiftedrectfft))
colormap(hsv)
```

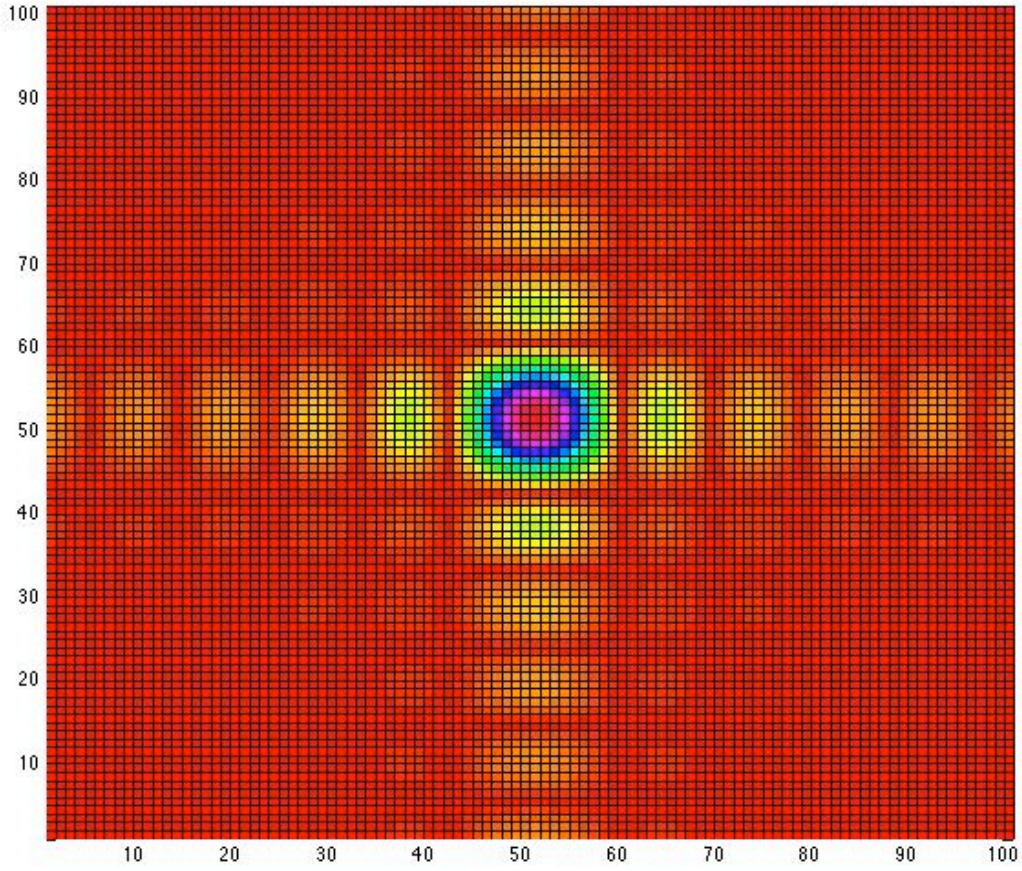

Figure 5: PCOLOR image of "diffraction pattern"

So now that the pre and post processing of an image and the information delivered by the fft2 function is understood, several common 2•D functions and their Fourier Transforms can be explored. The first function is a cosinusoidal wave defined by the following MATLAB commands.

```
x=1:101;y=x;
[X,Y]=meshgrid(x,y);
fx=0.03;fy=fx;
twodcos = cos(2*pi*fx*X + 2*pi*fy*Y);
surf(twodcos)
```

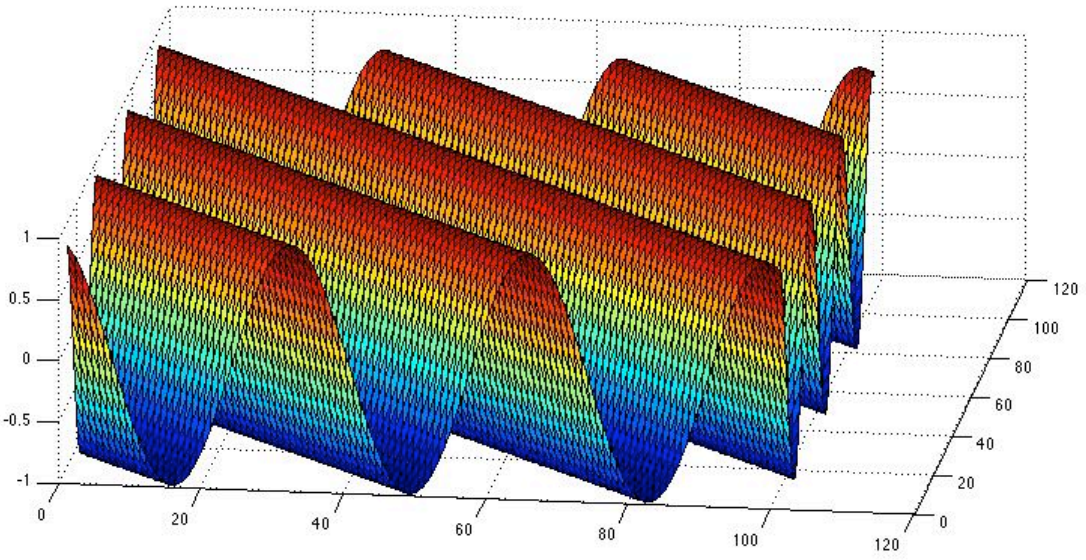

Figure 6: A cosinusoidal wave propagating at 45 degrees. For this propagation angle the values fx and fy in the function are equal.

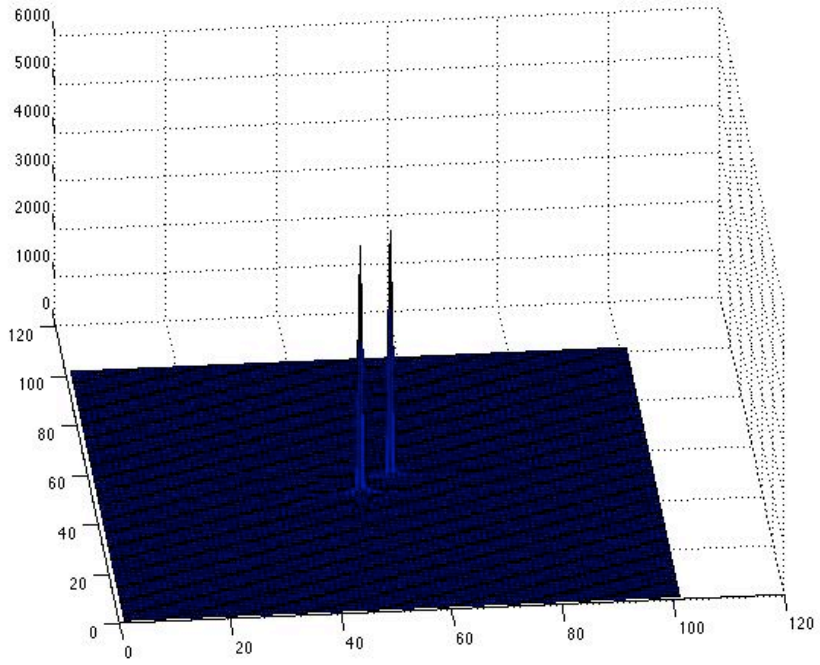

Figure 7: The Fourier Transform of the cosinusodial wave. Note that there are 2 delta functions representing the frequency of the cosinusodial function. Also note that these two delta functions are turned 45 degrees from axes.

Full codes are:

```
x=1:101;y=x;
[X,Y]=meshgrid(x,y);
fx=0.03;fy=fx;
twodcos=cos(2*pi*fx*X + 2*pi*fy*Y);
figure
surf(twodcos)
cosshifted=fftshift(twodcos);
cosfft=fft2(cosshifted);
shiftedcosfft=fftshift(cosfft);
figure
surf(abs(shiftedcosfft))
```

Another common function is the circular aperture that is commonly found in optics labs. By exploring its Fourier Transform some insight into its diffraction pattern can be gained.

Consider the MATLAB **fspecial** function

fspecial - Create predefined 2-D filter

Syntax

```
h = fspecial(type)
h = fspecial(type, parameters)
```

Description

h = fspecial(type) creates a two-dimensional filter h of the specified type.

Type	**Description**
'average'	Averaging filter
'disk'	Circular averaging filter (pillbox)
'gaussian'	Gaussian lowpass filter
'laplacian'	Approximates the 2-dimensional Laplacian operator
'log'	Laplacian of Gaussian filter
'motion'	Approximates the linear motion of a camera
'prewitt'	Prewitt horizontal edge-emphasizing filter
'sobel'	Sobel horizontal edge-emphasizing filter
'unsharp'	Unsharp contrast enhancement filter

h = fspecial('disk', radius) returns a circular averaging filter

(pillbox) within the square matrix of side 2*radius+1. The default radius is 5

Script circfft1.m below is defined by using the fspecial('disk',radius) command. This command softens the edges of the function, alleviating the discrete nature the circular would otherwise demonstrate. Discretely defined circles tend to have edges that are jagged. These jagged edges introduce frequency components that would not be present in a real circular aperture. These frequencies act as noise in the pattern and corrupt the frequency space diagram. To avoid this problem the 'soft' edged circ should be used.

The Cooley-Tukey algorithm assumes that the image is repeating in both the x and y directions. To bring about a more interpretable result this assumption must be avoided. The easiest solution is to simply pad the image with zeros increasing the period of the repeating pattern. The Script circfft1.m shown below is the same size has had zeros padded around it.

Now the Fourier Transform of the padded circular aperture can be taken. The frequency space image now resembles an Airy pattern.

```
% script circfft1.m
rect=zeros(201);
h=fspecial('disk',10)';
rect(90:110,90:110)=40000*h;
figure
pcolor(rect)
axis square
figure
image(rect)
axis square
colormap(gray)
circshifted=fftshift(rect);
circfft=fft2(circshifted);
shiftedcircfft=fftshift(circfft);
figure
surf(abs(shiftedcircfft))
figure
pcolor(abs(shiftedcircfft))
axis square
colormap(jet)
```

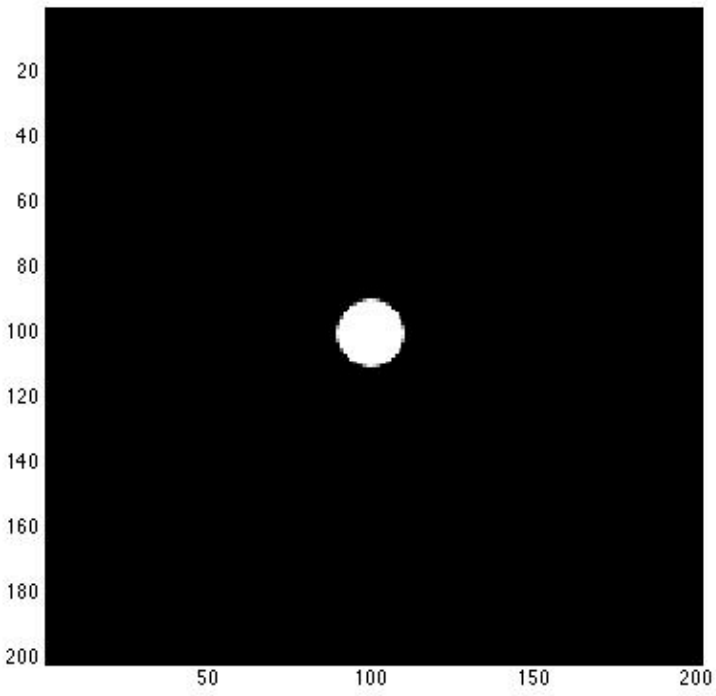

Figure 8: The circular aperture approximation

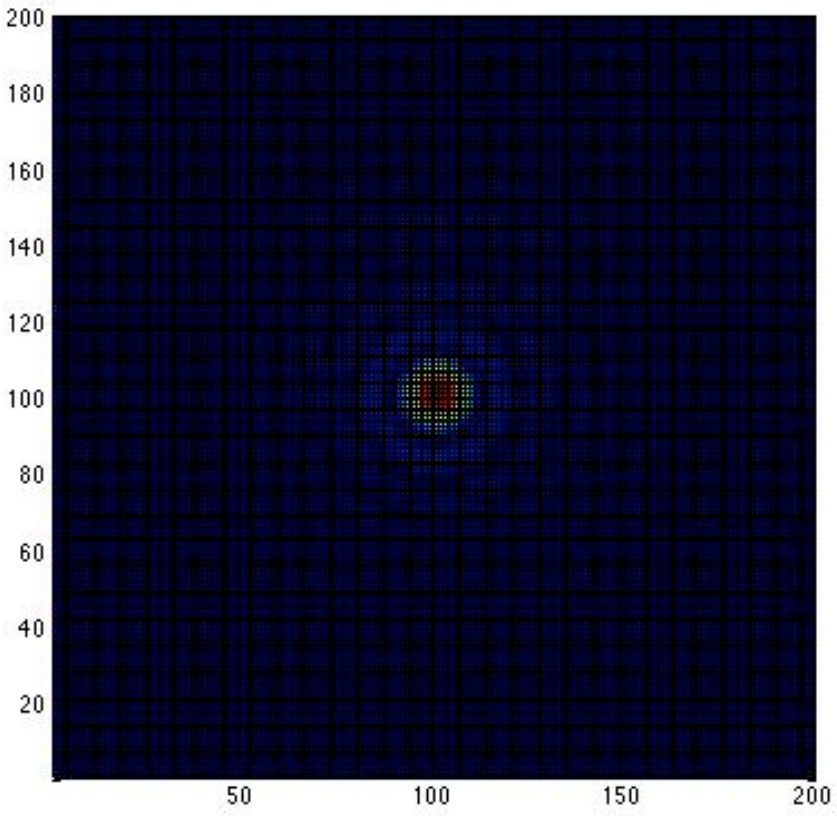

Figure 9: The Airy pattern = diffraction pattern

IV. CONCLUSIONS

An accurate 2-D Fourier Transform of a given image can be determined using MATLAB's fft2 command. Pre and Post processing of the image matrix through the use of fftshift makes the interpretation of the fft2 results easier. In the case of circularly symmetric functions, zero padding may be required bring about the desired results. This is due to the fft2's assumption that the image is infinitely periodic in both x and y.