Two-dimensional Fourier Transforms

6.1 Introduction

In this lab we will use two-dimensional Fourier transforms to simulate a famous historic experiment, the Young's double-slit experiment, devised by Thomas Young in 1801. This was the first demonstration of the wave nature of light, where the diffraction of light upon passing through two narrow slits formed interference fringes on a distant screen. Two-dimensional Fourier transforms are closely related to the phenomenon of diffraction in optics. The

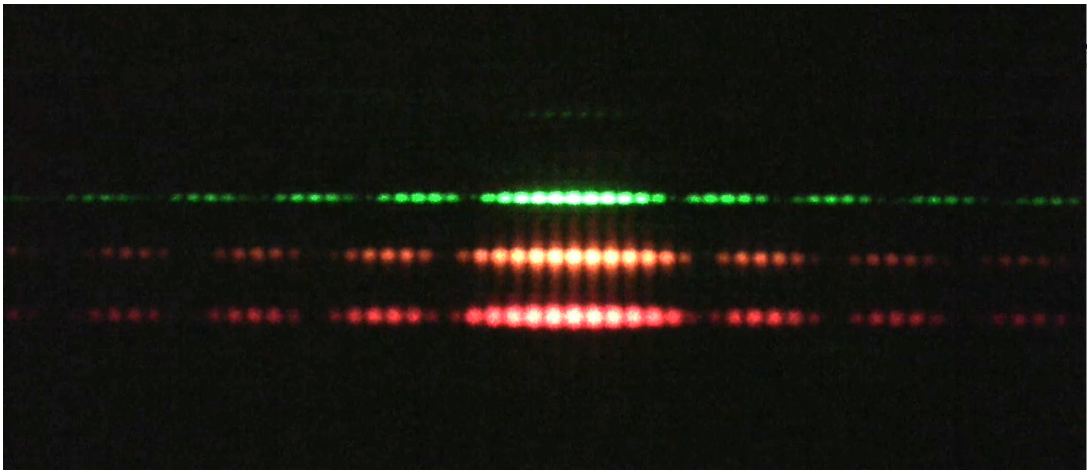


Figure 6.1: Interference pattern formed by shining lasers of different wavelengths through a double-slit aperture (source: `http://tsgphysics.mit.edu`).

diffraction pattern produced on a distant screen is the 2D Fourier transform of the aperture. The size and shape of the aperture directly influence the size and shape of the diffraction pattern in a very important way, which you will investigate in this lab. Along the way you will learn how to handle 2D arrays in MATLAB, and also several tips and tricks for visualising them.

6.2 Lab objectives

By the end of this lab session you should be able to:

1. Use **`fft2`** to perform DFTs on two-dimensional data and be familiar with its output format.
2. Know the basics of 2D array manipulation in MATLAB.
3. Know how to simulate Young's double-slit experiment in MATLAB.
4. Understand how the size/shape of a function and the size/shape of its Fourier transform are related.

If you still don't know how to do any of these things once you have completed the lab, please ask your tutor for help. You should also use these lists of objectives when you are revising for the end of semester exam.

6.3 Walkthrough: 2D arrays in MATLAB

This lab will make use of MATLAB's powerful capabilities for generating and manipulating 2D arrays. Below is a brief introduction to some of the functions and operators we will be using.

6.3.1 Creation

There are many ways to create 2D arrays. One way is using the MATLAB function **`zeros`**, which produces an array of zeroes. The integers passed as arguments to **`zeros`** correspond to the desired dimensions of the output array. The first number passed is the number of rows, the second number the number of columns, and so on. For example, to produce a 2D array **`A`** of zeroes with 3 rows and 5 columns, do

```
>> A = zeros(3,5)
A =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Another way to create 2D arrays is using the **`meshgrid`** function. This is very useful for creating arrays that represent the coordinates of some other array, which might contain the values of measurements at those coordinates. For example, if you had a 2D grid of measurements taken at coordinates **`x = [0.1,0.2,0.3,0.4]`** and **`y = [0.02,0.04,0.06]`** (i.e. each value of **`x`** for each value of **`y`**), then you could form the 2D coordinate grids as follows:

```
>> x = [0.1, 0.2, 0.3, 0.4];
>> y = [0.02, 0.04, 0.06];
>> [Mx, My] = meshgrid(x,y)
Mx =
    0.1000    0.2000    0.3000    0.4000
    0.1000    0.2000    0.3000    0.4000
    0.1000    0.2000    0.3000    0.4000

My =
    0.0200    0.0200    0.0200    0.0200
    0.0400    0.0400    0.0400    0.0400
    0.0600    0.0600    0.0600    0.0600
```

6.3.2 Manipulation

If two arrays **`A`** and **`B`** have the same dimensions, you can perform element-wise addition by using the **`+`** operator, i.e. the element-wise sum of **`A`** and **`B`** is **`A+B`**. The resultant array has the same dimensions as **`A`** and **`B`**.

You can reference single elements in 2D arrays by using parentheses and indexing the entries by two integers, the first corresponding to the row and the second corresponding to the column. For example, the value of the entry in the third row and second column of an array **`C`** is **`C(3,2)`**. To reference a range of entries, use the colon (**`:`**) operator. For example, to output an array containing just the entries between the second to sixth rows in the fifth column of an array **`D`**, type **`D(2:6,5)`**. Note that the slice notation is inclusive, i.e. **`2:6`** will retrieve the second and sixth rows, as well as everything in between. The resultant array in this example will have 5 rows and 1 column.

6.4 Walkthrough: 2D Fourier transforms

Multi-dimensional Fourier transforms are what you get when you apply one-dimensional Fourier transforms successively to each dimension in turn. If you have a function $z(x, y)$ defined along $-\infty < x, y < +\infty$, where x and y are two independent coordinates (they could be two spatial dimensions, or one spatial and one time dimension, etc.), then its two-dimensional Fourier transform is given by

$$Z(f_x, f_y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} z(x, y)\, \mathrm{e}^{-i2\pi(f_x x + f_y y)}\, \mathrm{d}x\, \mathrm{d}y\,, \tag{6.1}$$

where f_x and f_y are the frequencies corresponding to the x and y coordinates, respectively.

The MATLAB function for performing the two-dimensional Fourier transform is **`fft2`**, which is really just two

nested **fft**s in disguise: given an input 2D array **x**, the result of **fft2(x)** is the same as **fft(fft(x).').'**, where the **.'** operator in MATLAB transposes the array (i.e. flips rows and columns). Note that when **fft** is passed a 2D array rather than a vector, it performs a one-dimensional DFT along each individual column.

Just like for the **fft** function we used in the previous lab, the frequencies corresponding to the elements of the array output by **fft2** are in a semi-jumbled order, with the negative frequencies occurring after the positive frequencies along each axis. To re-order the array such that all frequencies are in ascending order along each axis, we can use the same **fftshift** function from before, which works just as well on multi-dimensional arrays as it does on vectors. The same odd/even rules for **fft** stated in the previous lab apply for **fft2**, now separately for each dimension. The effect of applying **fftshift** to a 2D array is illustrated in Fig. 6.2.

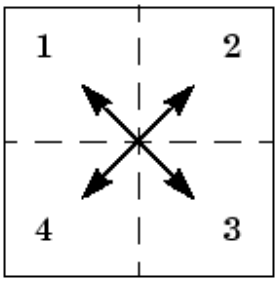


Figure 6.2: What the **fftshift** function does to a 2D array. Credit: MATLAB documentation.

6.5 Activity: Light on a rectangular aperture

When parallel wavefronts impinge upon an aperture, each point within the aperture acts as a source of secondary wavelets, in accordance with the Huygens-Fresnel principle. These wavelets constructively and destructively interfere, producing a series of bright and dark regions on a distant screen. When the screen is sufficiently distant, namely when the distance-to-aperture-size ratio greatly exceeds the aperture-size-to-wavelength ratio, we are in the *Fraunhofer regime* and the intensity pattern on the screen is proportional to the squared modulus of the 2D Fourier transform of the aperture.

An aperture can be simulated in MATLAB as a 2D array, by putting a 1 at every point where light can come through and a 0 everywhere else. To compute the Fraunhofer diffraction pattern, we then Fourier transform the the 2D array representing the aperture using **fft2** and plot the squared modulus of the result. (You might recall from lectures that although the electric field E of the light waves is what interferes to produce the diffraction pattern, it is the intensity $|E|^2$ that is actually observed).

In this section we will start off by simulating diffraction for a simple rectangular aperture, and later upgrade this to two slits for Young's double-slit experiment.

Question 1
Write a function **rect = rect_aperture(r1, c1, r2, c2)** that constructs a rectangular aperture on a 512×1024 grid, where the top-left corner of the rectangle has row, column coordinates **r1, c1** and the bottom-right corner has coordinates **r2, c2**.

Hint: you can use array slice notation of the form **A(r1:r2,c1:c2)** *to select the rectangular region bounded between rows* **r1** *and* **r2**, *and columns* **c1** *and* **c2** *(inclusive) within the array* **A**.

Test out this function by using it to generate a rectangular aperture 20 pixels high and 100 pixels wide. The top left corner should be at row, column coordinates of (247, 463). This corresponds to the rectangle being roughly at the centre of the grid.

Display your result using **imshow(rect)** with **axis on** to check the rectangle appears at the right location.

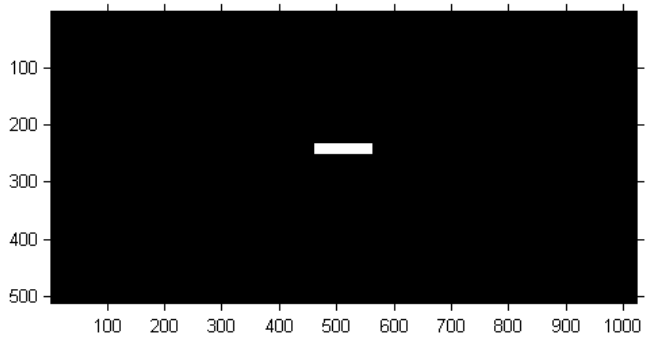
The **imshow** function is handy for visualising 2D arrays, but note that it shades the output plot assuming that the data lie between the range 0 and 1. Any values below 0 get assigned to black, and any above 1 get assigned to white. To improve the contrast when displaying arbitrary data with **imshow**, the function **mat2gray** will linearly rescale an input array such that the minimum value is mapped to 0 and the maximum value is mapped to 1. We needed to use **imshow** together with **mat2gray** when plotting the Fourier transform in the above example because its amplitudes were outside the range 0 to 1.

In the absence of any other input arguments, **imshow** will label its axes with the pixel coordinates (e.g. if you simply executed the command **imshow(g)**). If you want to put your own values on the axes, e.g. the frequencies **Fx** and **Fy** of the 2D Fourier transform, you can specify these using the options **'XData'** and **'YData'** as demonstrated above.

Sketch your plot below.

```
function rect = rect_aperture(r1, c1, r2, c2)
  rect = zeros(512, 1024);
  rect(r1:r2, c1:c2) = 1;
end

>> rect = rect_aperture(247, 463, 247+19, 463+99);
>> figure; imshow(rect)
>> caxis(0,0.01)
>> axis on
```



Tutor note: Encourage use of figure in this lab so diffraction patterns may be easily compared.

Question 2

Compute and plot the squared modulus of the 2D Fourier transform of the array you constructed in Question 1, using MATLAB's **fft2** and **fftshift** functions. Use the **mat2gray** function to rescale the amplitudes between 0 and 1, so that **imshow** plots this properly:

```
imshow(may2gray(abs(rect).^2))
```

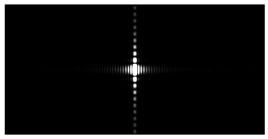
Note: It is not necessary to include the lines of code to label the axes correctly with frequency in Questions 2 to 8.

Plotting tip: To adjust the brightness scale on the output plot, the command

```
>> colormapeditor
```

brings up a GUI that allows you to make instant changes. Change 'Color data max' from 1.0 to 0.01 to enhance the features of the pattern. If you like, you can select between a range of inbuilt colour schemes (click on *Tools* → *Standard Colormaps*). Clicking and dragging on the colour bar allows you to stretch or compress certain ranges, and you can even add your own colours.

```
>> R = fftshift(fft2(rect));
>> imshow(mat2gray(abs(R).^2))
```

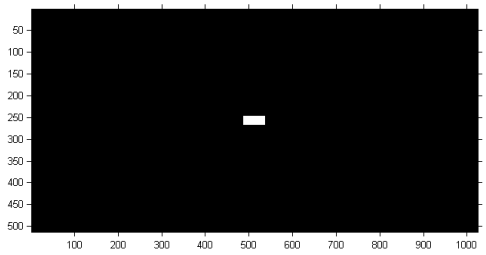


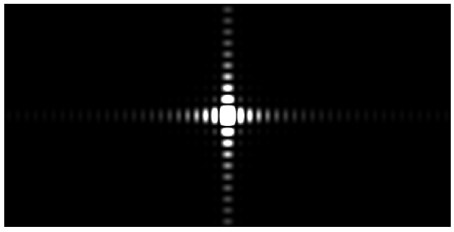
Tutor note: make sure students change the intensity scale of the plot, as described above, otherwise they won't be able to see any of the features.

Question 3

Now let's see what happens when we shrink the horizontal width of the aperture by a factor of 2, such that it now has 50 columns and its top-left corner is located at row, column coordinates of (247,487). Compute and plot the squared modulus of its 2D Fourier transform. How does this differ from the diffraction pattern in the previous question?

The diffraction pattern stretches by roughly a factor of 2 in the horizontal direction, compared to the diffraction pattern in Question 2.





Question 4

Based on your answer to Question 3, what can you say about the relationship between the characteristic size of the aperture and that of the diffraction pattern (i.e. Fourier transform)? Use this to explain the relative widths of the diffraction pattern in Question 2 between the horizontal and vertical directions.

The characteristic width of the aperture and that of its Fourier transforms are inversely related. If the aperture is narrow along a certain direction then its Fourier transform will be wide along that direction. If the width of the aperture increases by a certain factor, the width of its Fourier transform decreases by that factor and vice versa. We made no change to the vertical width of the aperture between Questions 2 and 3, and so there was no change to the vertical scale of the diffraction pattern. In Question 2 the diffraction pattern was wider in the vertical direction than it was in the horizontal direction, because the aperture was narrower in the vertical direction than the horizontal direction.

Checkpoint 1:

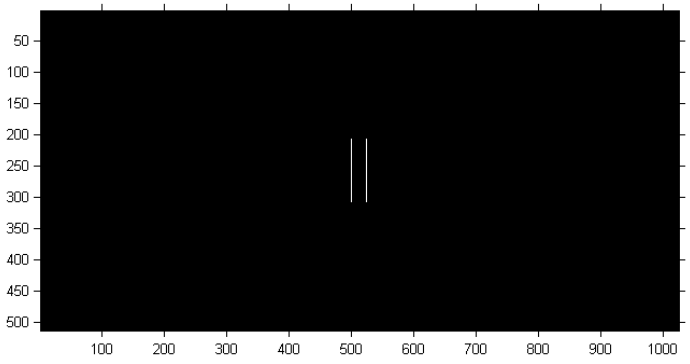
6.6 Activity: Young's double-slit experiment

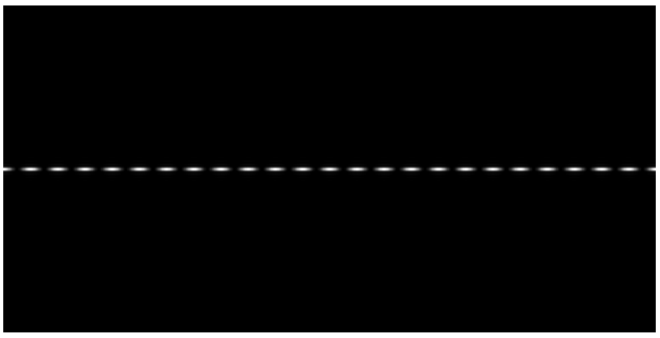
We are now ready to simulate Young's double-slit experiment. We will construct the two slits as two thin rectangular apertures with the long axis oriented in the vertical direction, and compute the diffraction pattern using the **`fft2`** function. The squared modulus of the result corresponds to the intensity pattern one would see on a distant screen if light were to be shone through the two slits.

In class, you would have treated Young's double-slit experiment as a one-dimensional problem, assuming that the slits were infinite in length. Here we will simulate this on a 2D grid, where both slits have a finite length. The effect of giving the slits a finite length is to broaden the diffraction pattern along the direction in which the slits are oriented. This bears closer resemblance to what would actually occur in reality.

Question 5
Using the function you wrote for Question 1, construct an aperture with two long, narrow vertical slits of width 1 and height 100, with their top edges aligned with row 207. Place one of them at column 500 and the other at column 524. You can do this by constructing each slit separately, then adding the two arrays together using MATLAB's element-wise addition operation. Compute and plot the squared modulus of the 2D Fourier transform of this aperture.

```
>> slit1 = rect_aperture(207, 500, 207+99, 500);
>> slit2 = rect_aperture(207, 524, 207+99, 524);
>> aperture = slit1 + slit2;
>> A = fftshift(fft2(aperture));
>> imshow(mat2gray(abs(A).^2))
```

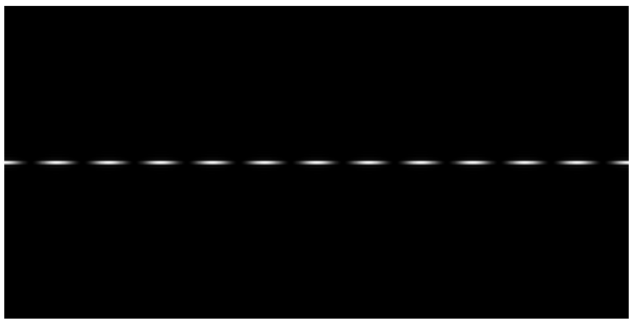




Question 6
Now move the slits closer together by placing the first one at column 506 and the second one at column 518 (do not change the size of each slit). How does the diffraction pattern change?

```
slit1 = rect_aperture(207, 506, 207+99, 506);
slit2 = rect_aperture(207, 518, 207+99, 518);
aperture = slit1 + slit2;
A = fftshift(fft2(aperture));
imshow(mat2gray(abs(A).^2))
```

The pattern widens by a factor of two, i.e. the spacing between adjacent peaks doubles compared to before.

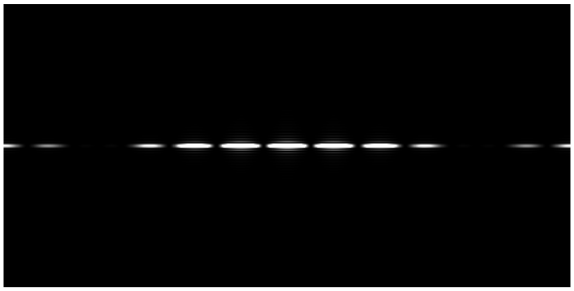


Question 7
Slits of width 1 can be thought of as the numeric equivalent of δ-functions (infinitesimally thin spikes). We are now going to see what happens if the slits have a finite width. Construct a double-slit aperture again, but this time give the slits a width of 3 pixels. The top-left corners of the two slits should now be at row, column coordinates of (207,505) and (207,517). Compute and plot the squared modulus of its 2D Fourier transform. How does this differ from the pattern in the previous question?

Some features may be difficult to see with the default brightness scaling. Bring up the `colormapeditor` GUI and change 'Color data max' from 1.0 to 0.1 to enhance the contrast.

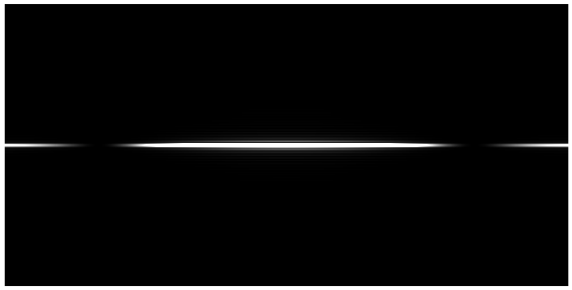
```
slit1 = rect_aperture(207, 505, 207+99, 505+2);
slit2 = rect_aperture(207, 517, 207+99, 517+2);
aperture = slit1 + slit2;
A = fftshift(fft2(aperture));
imshow(mat2gray(abs(A).^2))
```

We still see a pattern of peaks, with the same spacing as before and in the same positions, but the peaks near the centre are brightest and their brightness changes with horizontal location. They reach minimum brightness at around 20% from the edges of the screen, and then become brighter again as you move further out.



Question 8 (PHYS2911 and PHYS2921)
Compute and plot the squared modulus of the 2D Fourier transform for a single slit of width 3 pixels, with the top-left corner positioned at row, column coordinates of (207,512). You may want to set 'Color data max' to 0.1 again to enhance the contrast. Can you guess how this pattern combines with the pattern from Question 6 to produce the one in Question 7? The way in which these two patterns combine is known as *convolution*.

```
slit = rect_aperture(207, 512, 207+99, 512+2);
S = fftshift(fft2(slit));
imshow(mat2gray(abs(S).^2))
```



The pattern in Question 7 is the product of this pattern and the one in Question 6. The double-slit pattern where the two slits have finite width is the double-slit pattern for infinitesimally thin slits, modulated by the pattern for a single slit of finite width.

6.7 Activity: Plane waves

A 2D plane wave is a function that oscillates sinusoidally in one direction and is constant along the orthogonal direction. It has the example functional form

$$z(x, y) = \cos(2\pi f_x x + 2\pi f_y y)\,, \tag{6.2}$$

where f_x and f_y describe the projections of the oscillation frequency along the x and y axes. For example, a plane wave with $f_x = 1$ and $f_y = 2$ experiences one oscillation every unit of x and two oscillations for every unit of y. The lines of constant phase are given by $f_x x + f_y y$ = constant. These lines are slanted with respect to the x and y axes. The oscillation direction (the *wavevector*) is perpendicular to the lines of constant phase, and the oscillation frequency f is given by $f = \sqrt{f_x^2 + f_y^2}$ (see Fig. 6.3). These are the two-dimensional analogues of the one-dimensional sinusoids you saw in the previous lab.

The following lines of code show how to construct a plane wave with these parameters and display it as a 3D surface plot using the inbuilt MATLAB function **`surf`**.

```
x = 0.02:0.02:2;
y = 0.02:0.02:2;
[Mx, My] = meshgrid(x,y);
fx = 1;
fy = 2;
z = cos(2*pi*fx*Mx + 2*pi*fy*My);
surf(x, y, z, 'LineStyle', 'none')
xlabel('x'); ylabel('y'); zlabel('z')
axis([0 2 0 2 -3 3])
```

The surface plot generated by these lines of code is shown in Fig. 6.3.

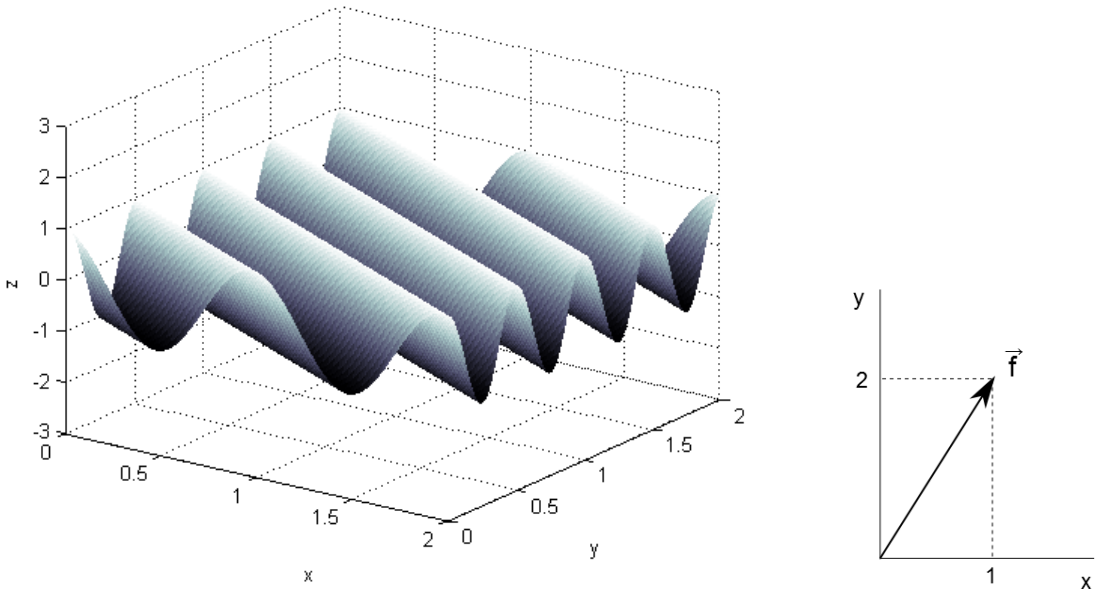


Figure 6.3: Left: Surface plot of a 2D plane wave, with a frequency of $f = \sqrt{f_x^2 + f_y^2} = \sqrt{5}$. Right: The corresponding wavevector, which lies in the x, y-plane.

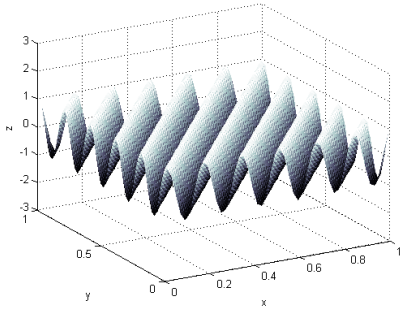
Question 9 (PHYS2011 and PHYS2911)

This question is optional for PHYS2921

Construct a 2D plane wave with $f_x = 6$ and $f_y = -5$, over the the x-range `x = 0.02:0.02:1` and y-range `y = 0.02:0.02:1`. Display this as a surface plot like in the example.

Plotting tip: The command **`axis`** allows you to set the display range along the different axes of a plot. For a three-dimensional plot such as a surface plot with axes x, y and z, you can set the display range by passing **`axis`** a vector of the form `[xmin xmax ymin ymax zmin zmax]`. For this particular example, a suitable display range for the plane wave turns out to be given by `axis([0 1 0 1 -3 3])`.

```
fx = 6; fy = -5;
x = 0.02:0.02:1;
y = 0.02:0.02:1;
[Mx, My] = meshgrid(x,y);
z = cos(2*pi*fx*Mx + 2*pi*fy*My);
figure; surf(x,y,z, 'LineStyle', 'none')
xlabel('x'); ylabel('y'); zlabel('z')
axis([0 1 0 1 -3 3])
```



Question 10 (PHYS2011)
Considering the 2D plane wave in Question 9, compute the squared modulus of its 2D Fourier transform, and display this as a surface plot. Remember to use **`fftshift`**. *To expand the data to fill all the plot space, use the command* **`axis tight`**

```
Z = fftshift(fft2(z));
figure; surf(x, y, abs(Z).^2)
xlabel('x'); ylabel('y'); zlabel('|Z|^2')
axis tight
```

Tutor: the plot is similar to the one for Question 10 below (PHYS2911), except that the axes are not labelled properly. The peaks' coordinates are: $(0.42, 0.66)$**,** $(0.66, 0.42)$**,)**

Question 10 (PHYS2911)

This question is optional for PHYS2921

Considering the 2D plane wave in Question 9, compute the squared modulus of its 2D Fourier transform, and display this as a surface plot. Remember to use **`fftshift`**, and label the axes appropriately. Check that the peaks appear at the correct frequencies. Recall from Lab 5 how to calculate frequencies:

$$Nq = \frac{1}{(2 * dx)}$$

; if the number of elements is odd, **`f = -Nq+0.5/L : 1/L : Nq-0.5/L`**;
if the number of elements is even, **`f = -Nq : 1/L : Nq-1.L`**.

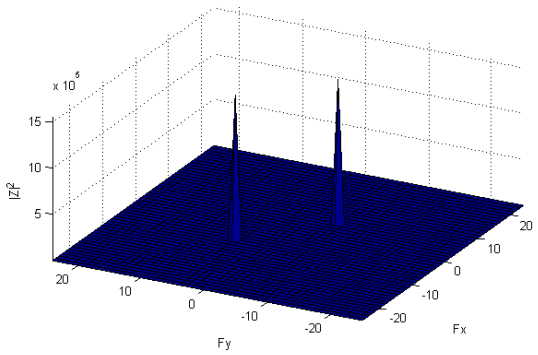
Tip 1: You can use the Data Cursor tool to query the coordinates of the peaks (the button for this has a black '+' and a pale yellow box – you can find this on every MATLAB figure panel).

Tip 2: To change the transparency of the surface plot, invoke the **`'FaceAlpha'`** option with a number between 0 (transparent) and 1 (opaque) when you call **`surf`**, e.g.

```
>> surf(x,y,z, 'FaceAlpha', 0.5)
```

to produce 50% transparency.

```
Z = fftshift(fft2(z));
Nq = 1/(2*0.02);
Fx = -Nq : Nq-1;
Fy = -Nq : Nq-1;
figure; surf(Fx, Fy, abs(Z).^2)
xlabel('Fx'); ylabel('Fy'); zlabel('|Z|^2')
axis tight
```



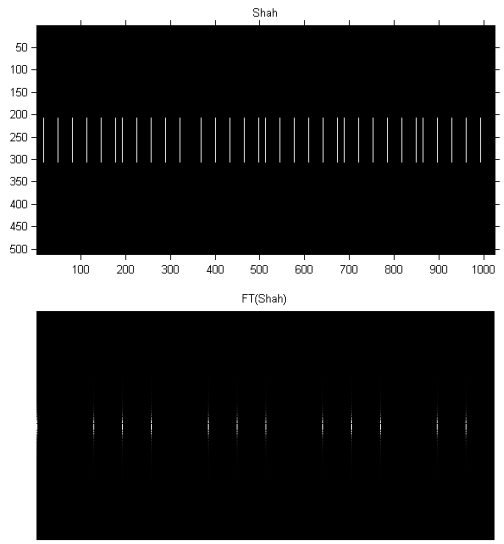
6.8 Diffraction grating PHYS2921

Questions 11 and 12 are optional for PHYS2011 and PHYS2911

Question 11 (PHYS2921)
The Shah function is an infinite series of evenly-spaced spikes. Using the function you wrote in Question 1 (or otherwise), construct a series of evenly-spaced slits tiled along the whole horizontal length of the grid, with a spacing of 16 pixels. Compute and display the squared modulus of the Fourier transform of this pattern, using **`imshow`** and **`mat2gray`**. (Although the array isn't infinite, it does reasonably well in replicating the actual Shah function and its Fourier transform.) To improve the contrast, bring up the **`colormapeditor`** and set 'Color data max' to 0.01.

Aside: As you may have seen in class, an aperture constructed from a long series of regularly-spaced slits like this is called a *diffraction grating*.

```
aperture = zeros(512,1024);
for i = 1:16:1024
  aperture(207:207+99, i) = 1;
end
A = fftshift(fft2(aperture));
figure; imshow(mat2gray(abs(A).^2))
```



Tutor note: You may need to zoom into the plots to convince yourself that all the evenly-spaced lines are present, since these are quite thin.

Question 12 (PHYS2921)
What property do you notice the Shah function sharing in common with Gaussians, but not plane waves?

The property the Shah function shares in common with Gaussians but not plane waves is that its Fourier transform is a function of the same type, i.e. another Shah function. However, this property does not hold for plane waves, whose Fourier transforms are δ-function spikes and do not resemble the original function.

Checkpoint 2: