THE UNIVERSITY OF SYDNEY

LAB

SEVEN

Radio Interferometry

7.1 Introduction

Observations in astronomy are often limited by the effects of diffraction. The angular resolution of a telescope is of order λ/D, where λ is the observing wavelength and D is the diameter of the telescope (the Rayleigh criterion). This means that when λ is large, D must be large to achieve a good resolution. The extreme case of a diffraction-limited regime is radio astronomy, where the wavelengths (centimetres to metres) are comparable to the physical size of the telescope components themselves. This makes high-resolution imaging challenging at radio wavelengths. For example, if we were to build a radio telescope that had the same resolution as the Hubble Space Telescope (which operates at visible wavelengths), the radio telescope would need to be as big as the Earth!

Figure 7.1: The Australian Telescope Compact Array, a 6 antenna radio interferometer located in Narrabri, NSW. Only 5 antennae are visible in the image, the 6th is located 6 km away.

The birth of radio astronomy was marked by Karl Jansky's discovery of radio emission from the centre of the Milky Way, which he announced in 1933. Subsequently, the development of radio astronomy using radar equipment left over from World War II was led by groups from Cambridge University, and also the CSIRO here in Australia. A clever technique to achieve high angular resolutions without needing to build an unreasonably large telescope is *interferometry*, also called *synthesis imaging*. This was developed in 1946 by the Australian pioneers Lindsay McCready, Joseph Pawsey and Ruby Payne-Scott, the last of whom was a graduate of the University of Sydney. Radio interferometry uses Fourier transforms to combine the signals measured by many different radio receivers, to achieve the equivalent resolution of a much larger telescope with a diameter equal to the spacings of the smaller receivers. This has made high-resolution imaging possible at radio wavelengths. Thanks to these developments, Australia is now the world leader in radio astronomy.

In this lab, you will learn the basics of how radio interferometers produce images of objects in the sky.

7.2 Lab Objectives

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

1. Understand what a response function is and how it affects scientific measurements.
2. Use `fft2` and `ifft2` to perform two-dimensional convolution.
3. Understand the link between Fourier transforms and interferometry.
4. Simulate a multi-element interferometer.

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.

7.3 Walkthrough: Inverse Fourier transforms

Given the Fourier transform of a function, you can perform the *inverse Fourier transform* to recover the original function. For example, if $Y(f)$ is the Fourier transform of a function $y(x)$, (see equation 5.1), then the latter is equal to

$$y(x) = \int_{-\infty}^{+\infty} Y(f)\, \mathrm{e}^{+i2\pi f x} df\,. \tag{7.1}$$

For one-dimensional arrays in MATLAB, the function to do this is **`ifft`**. This is an exact inverse of **`fft`**, in that any input **`y`** should be equal to **`ifft(fft(y))`**. If the original signal is real, then taking the inverse Fourier transform should recover the purely real input, even though **`fft(y)`** may be complex. However, internal roundoff error can sometimes lead to the appearance of small imaginary components in the inverse Fourier transformed signal. These can be suppressed by invoking the **`ifft`** function with the **`'symmetric'`** option, e.g.

```
y = ifft(Y, 'symmetric');
```

An important point to note is that **`ifft`** expects input of the format output by **`fft`**, i.e. with the strange frequency ordering. If you used **`fftshift`** to reorder the array output by **`fft`**, you need to put it back using **`fftshift`**'s inverse function, **`ifftshift`**, before you pass it to **`ifft`**. E.g. if you started off with some data **`y`**, the commands

```
Y = fftshift(fft(y));
x = ifft(ifftshift(Y), 'symmetric');
```

will recover a vector **`x`** that is identical to **`y`**. **`ifftshift`** works for arrays of any dimension, just like **`fftshift`**.

We can illustrate the inverse Fourier transform using the example of a sine wave with frequency $f = 3$, the same one you saw way back in Section 5.6. The code to generate and plot its Fourier transform is

```
dx = 0.01;
L = 2;
x = dx:dx:L;
f = 3;
Y = fftshift(fft(sin(2*pi*f*x)));
Nq = 0.5/dx;
freqs = -Nq : 1/L : Nq-1/L;
plot(freqs, abs(Y).^2)
```

producing the plot on the left in Fig. 7.2. Remember that the frequency vector (**`freqs`**) is define as

```
freqs=-Nq:1/L:Nq-1/L
```

for even numbers of sample and

```
freqs=-Nq+1/(2*L):1/L:Nq-1/(2*L)
```

for odd numbers of sample.

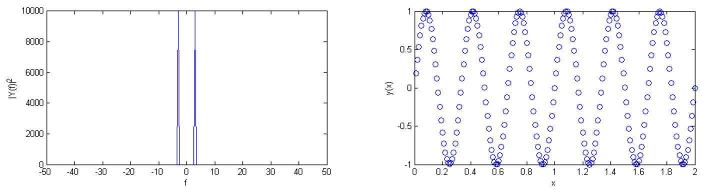


Figure 7.2: A pair of δ-function spikes at $f = 3$ (left), and their inverse Fourier transform (right).

The δ-function spikes at ± 3 tell us that this describes a signal containing a single frequency component with $f = 3$. Now see what happens when we compute and plot the inverse Fourier transform of **`Y`**:

```
y = ifft(ifftshift(Y), 'symmetric');
plot(x, y, 'o')
```

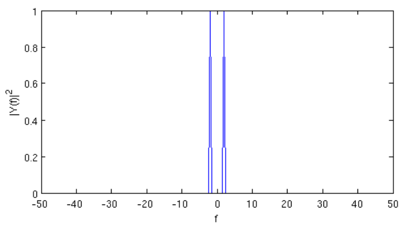
The resultant vector **`y`** is plotted in the right panel of Fig. 7.2. If you count the number of oscillations per unit length, you'll see that this is indeed a sine wave of frequency 3.

Question 1

In the example we just saw, we obtained the pair of δ-function spikes by initially Fourier transforming a sine wave. An alternative is to construct the δ-function spikes directly. Refer to the code on page 2 and create a vector `freqs`, then construct a vector `Y` of zeroes of the same size and put a 1 where the frequencies are ± 2. This corresponds to a pair of δ-function spikes at $f = 2$. Sketch your plot below.

Hint:
You can use MATLAB's logical indexing notation to achieve this. For example, `A(B == -2 | B == 2) = 1` *performs the operation "wherever the entries of* `B` *equal ± 2, change the corresponding entries of* `A` *to 1". The arrays* `A` *and* `B` *must have the same dimensions.*

```
dx = 0.01;
L = 2;
x = dx:dx:L;
Nq = 0.5/dx;
freqs = -Nq : 1/L : Nq-1/L;
Y = zeros(size(freqs));
Y(freqs == -2 | freqs == 2) = 1;
```

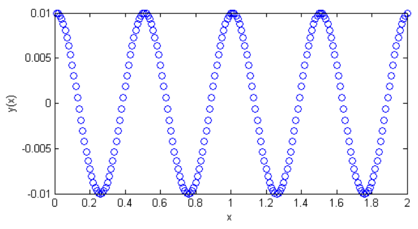


Question 2

Compute the inverse Fourier transform, and sketch the result below. By counting the number of oscillations per unit length, what is the frequency of this wave? Does this match the input?

```
y = ifft(ifftshift(Y), 'symmetric');
plot(x, y, 'o')
```

The waveform undergoes two oscillations per unit length, so the frequency is 2. This matches the input.



Tutor note: This is a cosine rather than a sine wave because the way we constructed the amplitudes implicitly sets all phases to zero (only cosine terms present).

7.4 Walkthrough: Convolution

The concept of convolution is of fundamental importance to astronomical imaging. Mathematically, convolution is an operation denoted by the symbol $*$ that in the one-dimensional case takes as input two functions, f and g, and outputs

$$[f * g](x) = \int_{-\infty}^{+\infty} f(\xi)g(x-\xi)\,\mathrm{d}\xi\,. \tag{7.2}$$

The Convolution Theorem states that

$$\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)\,, \tag{7.3}$$

where $\mathcal{F}$ means "Fourier transform of". This means that convolution can be performed by multiplying the Fourier transforms of f and g together, and then inverse Fourier transforming the result. Although there is a discrete version of Eq. (7.2) that is implemented in MATLAB with an inbuilt function **`conv`**, in this lab we will perform convolution based on Eq. (7.3), i.e. using MATLAB's FFT functions.

Qualitatively, the effect of convolution is to "smear" the two functions f and g together, so that the end result has features from both[1]. To demonstrate this, let's convolve a square pulse with a triangular pulse, as shown below.

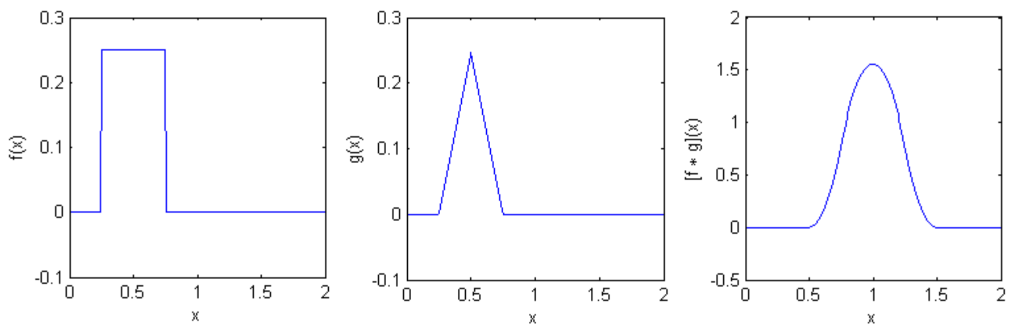


Figure 7.3: When a square pulse (left) is convolved with a triangular pulse (middle), the result is a function that looks somewhat in-between the two (right).

Firstly, we can create and plot a square pulse with the following MATLAB code:

```
x = linspace(0,2,200);
box = zeros(200,1);
box(x > 0.25 & x < 0.75) = 0.25;
plot(x, box)
```

Then we can create a triangular pulse with similar code:

```
triangle = zeros(200,1);
triangle(x > 0.25 & x < 0.5) = x(x > 0.25 & x < 0.5) - 0.25;
triangle(x > 0.5 & x < 0.75) = 0.75 - x(x > 0.5 & x < 0.75);
plot(x, triangle)
```

Try plotting these and make sure they look like you expect. Now to convolve the two functions we need to Fourier transform them, multiply them together, and then take the inverse transform:

```
fftconv = fft(box) .* fft(triangle);
boxtriangle = ifft(fftconv, 'symmetric');
plot(x, boxtriangle)
```

Remember that the `.*` operator does element-wise multiplication for two arrays that have the same dimensions. This process is illustrated in Fig. 7.3, where you can see that the final result looks like something in-between a square pulse and triangular pulse.

[1]See `http://en.wikipedia.org/wiki/Convolution` for an animation of this effect.

The two-dimensional analogue of `ifft` is `ifft2`, where for a given 2D array `x`, the result of `ifft2(fft2(x))` recovers `x` (up to the effects of roundoff error). To guard against roundoff error, use the the option `'symmetric'` when calling `ifft2`, just as for `ifft`.

Convolution is important in astronomy because the image formed by a telescope is not the true brightness distribution in the sky – it is the brightness distribution convolved with something called the *response function* of the telescope. The response function is the brightness distribution measured by the telescope in response to a bright point-like object, and can be calculated as the inverse Fourier transform of the telescope aperture.

In this lab, we will see how the moon appears through a telescope, by using the MATLAB's demo image fo the moon.

We shall proceed as follows:

1. Start with the image of the moon (*moon*)
2. Find its Fourier Transform (M).
3. Find the response function (f_{resp}) of the circular aperture (C), using the Inverse Fourier Transform.
4. Find the image formed by the telescope by the convolution of moon and f_{resp}, or rather, by calculating the inverse fourier transform of the product M and C.

7.5 Activity: The Moon through a circular aperture

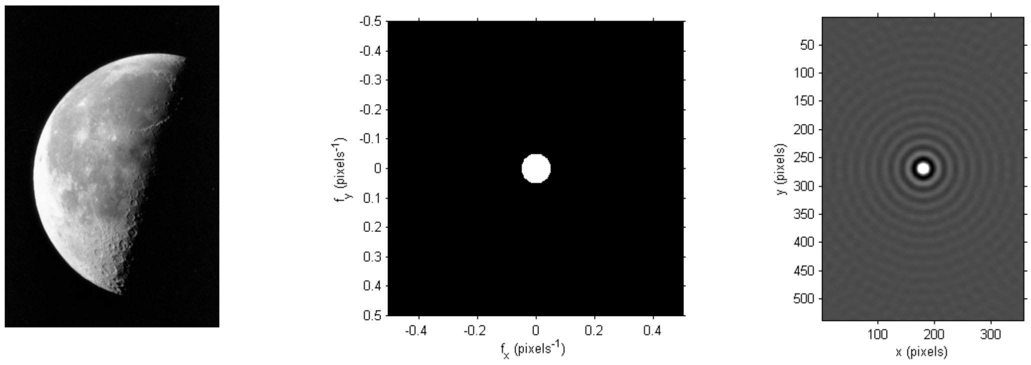


Figure 7.4: Left: image of the Moon (credit: Michael Myers). Middle: circular aperture . Right: Response function of the circualr aperture.

Step 1: Load and display the image

You can load and display the MATLAB's demo image of the Moon (left panel in Fig. 7.4) using:

```
moon = imread('moon.tif');
imshow(moon)
```

Step 2: Fourier Transform and Frequency vectors

Our goal is to calculate the Fourier Transform of the image, but first we need to calculate the frequency vectors (`fx` and `fy`) as we did in Lab 5 and 6.

Question 3

In Lab 5, you constructed frequency vectors for one-dimensional data. For two-dimensional data you will need to construct two frequency vectors, one for each dimension. To work out the two vectors, you need apply those rules for each dimension separately, because the input array will in general have different resolutions and ranges along its different dimensions.

You can work out the dimensions of the Moon image using `[Ly, Lx] = size(moon)` where `Lx` and `Ly` are the horizontal and vertical dimensions, respectively, in pixel coordinates.

Based on these values, and using the rules from Lab 5, write down the commands you would use to construct the frequency vectors `fx` and `fy` corresponding to the x and y dimensions. You can assume that each pixel represents one unit of length, implying that the Nyquist frequencies along each dimension, `Nqx` and `Nqy`, are both 0.5.

```
[Ly, Lx] = size(moon);
Nqx = 0.5; Nqy = 0.5;
fx = -Nqx : 1/Lx : Nqx-1/Lx;
fy = -Nqy+0.5/Ly : 1/Ly : Nqy-0.5/Ly;
```

Tutor note: Care Ly is first because MATLAB places rows or vertical axis first. The dimensions of the image are an even $\times$ odd number of elements, hence the different expressions for the frequency vectors.

Step 3: Calculate the Fourier Transform of the image (`moon`)

Question 4 Compute and plot the squared modulus of the 2D Fourier transform of the Moon image, labelling the frequency axes with the vectors `fx` and `fy` that you constructed in Question 3.

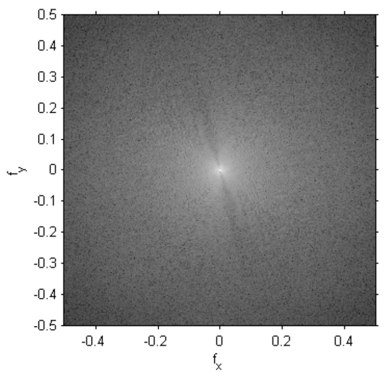
Plotting tip: The amplitudes of the Fourier transform span a very wide range of scales, so it may help to plot the logarithm of the values instead. You can use either the MATLAB function `log` for the natural logarithm or `log10` for the base-10 logarithm.

A brief note about plotting 2D arrays: MATLAB displays these by default with the first row, column entry on the top-left. This means that the x-axis increases from left to right and the y-axis increases from top to bottom. If you want to flip the y-axis so that it matches the plotting convention most people are used to (y-values increasing upwards), use the command

```
set(gca, 'YDir', 'normal')
```

when the figure whose y-axis you want to flip is open.

```
M = fftshift(fft2(moon));
imshow(mat2gray(log10(abs(M).^2)), 'XData', fx, 'YData', fy)
axis on; xlabel('f_x'); ylabel('f_y')
set(gca, 'YDir', 'normal')
```



Step 4: Calculate the response function of the circular aperture A circular aperture with a radius of 0.05 in spatial frequency units is shown in the middle panel of Fig. 7.4. The MATLAB commands to construct this aperture are

```
[Mfx, Mfy] = meshgrid(fx, fy);
radius = 0.05;    % we are choosing radius to be 0.05
C = zeros(size(M));%where M is the FFT of the Moon's image
C(Mfx.^2 + Mfy.^2 < radius^2) = 1;
imshow(C, 'XData', fx, 'YData', fy)
```

where **fx** and **fy** are the vectors containing the frequency coordinates (see Question 3). The corresponding response function (the image of a point source due to diffraction by the aperture), which is the famous Airy disk pattern (Fig. 7.4, right) can be computed via:

```
%Response function
f_resp = ifft2(ifftshift(C), 'symmetric');
imshow(mat2gray(f_resp))
```

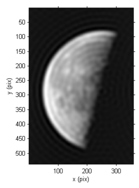
Take note that we need to use **ifftshift** before calling **ifft2**, because **ifft2** expects input of the format output by **fft2**, i.e. with the frequencies in the strange semi-jumbled order.

Note that the Fourier Transform of the response function for the circular aperture is the circular aperture itself

Step 5: Find the image formed by the telescope.

Question 5
The image of the Moon through the telescope is the convolution of source image (the Moon) and the response function (the Airy disk). You need to multiply the FFT of the Moon (see Question 4) and the FFT of the Airy disk (this is **C** -see above) and take the inverse 2D Fourier Transform using **ifft2**. Remember to invoke the **'symmetric'** option. Plot the resulting image using **imshow** and **mat2gray**, and comment on what you see.

```
M = fftshift(fft2(moon));
[Mfx, Mfy] = meshgrid(fx, fy);
radius = 0.05;
C = zeros(size(M));
C (Mfx.^2 + Mfy.^2 < radius^2) = 1;
mcircle = ifft2(ifftshift(M.*C), 'symmetric');
imshow(mat2gray(mcircle))
```



The resulting image is a blurry version of the original. Faint ring-like features are artefacts that come from the Airy disk response function.

Checkpoint 1:

7.6 Activity: The 2-element interferometer

The simplest radio interferometer comprises two receiving elements. The electric fields associated with radio waves arriving at the interferometer produce voltage fluctuations at the two receivers, and these signals are then correlated (multiplied together and averaged over time). A schematic of a 2-element interferometer is shown in Fig. 7.5 (left). If the path difference between the two receivers for an incoming wavefront is an integer number of wavelengths, constructive interference occurs and the correlated signal has maximum amplitude. If the path difference is a half-integer number of wavelengths, then there is destructive interference and the correlated signal drops to zero.

This is exactly the same principle that gives rise to the interference fringes in Young's double-slit experiment! In fact, the 2-element interferometer can be thought of as Young's double-slit experiment *in reverse*. The difference is that the "slits" are now radio receivers, and the electric field is being received rather than emitted.

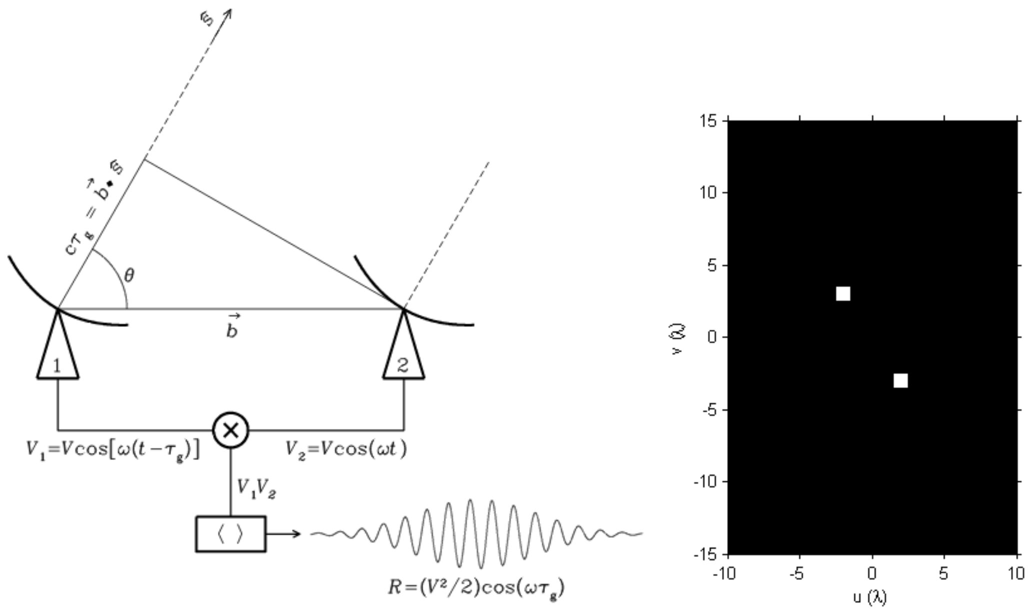


Figure 7.5: Left: A simple 2-element interferometer (source: `www.cv.nrao.edu/~sransom/web/Ch3.html`). Right: The u, v-coverage of a 2-element interferometer with a baseline vector of $\vec{b} = (2, -3)$.

We shall proceed as follows:

1. Start with the image of the moon
2. Find its Fourier Transform
3. Define the reponse function as the Inverse Fourier Transform of the *uv*-plane
4. Find the image formed by the radio interferometer by the convolution of the moon's image and the interferometer response function. As before, this corresponds to the Inverse Fourier Transform of the product between the Fourier Transform of the moon's image and the Fourier Transform of the telescope response function.

Steps 1 and 2

Load and display the image of the moon and calculate its Fourier Transform as before.

Step 3 A pair of radio receivers that are correlated together forms a *baseline*. In general, a radio interferometer comprises more than two receivers, and therefore more than one baseline. Every pair of receivers in an interferometric array measures one spatial frequency of the sky's brightness distribution. The spatial frequency f is related to the baseline b and observing wavelength λ by $f = b/\lambda$. In two dimensions, f and b are vectors with x and y components. For example, if the two receivers are at (1,0) m and (5,−6) m and the observing wavelength is $\lambda = 2\,$m, then the baseline vector is $\vec{b} = (5, -6) - (1, 0) = (4, -6)\,$m and the spatial frequency measured is

$\vec{f} = \vec{b}/\lambda = (2, -3)$ in units of λ. The "aperture" corresponding to this interferometer has two "holes", at $\pm\vec{f}$, and is shown in the right panel of Fig. 7.5. A plot like this is called a "u, v-coverage plot", where u and v refer to the horizontal and vertical spatial frequencies, respectively (here $u = \pm 2$ and $v = \mp 3$ in units of λ).

For simplicity, we are now going to rescale the frequency units such that each pixel in the 2D Fourier array corresponds to one unit of λ. The frequency vectors for our Moon example can then be constructed as

```
u = int32(-Lx/2 : Lx/2-1);
v = int32(-(Ly-1)/2 : (Ly-1)/2);
```

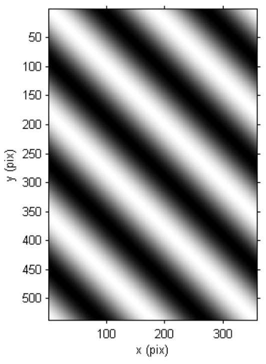
where `Lx` and `Ly` are the dimensions of the Fourier grid (equal to the dimensions of the Moon image). The MATLAB commands for generating the u, v-plot in Fig. 7.5 (right) are

```
[Mu,Mv] = meshgrid(u,v);
bx = 2; by = -3;
uvplane = zeros(size(M));
uvplane(Mu == bx & Mv == by) = 1;
uvplane(Mu == -bx & Mv == -by) = 1;
imshow(uvplane, 'XData', u, 'YData', v)
set(gca, 'YDir', 'normal')
```

This constructs a 2D array of zeroes, and puts a 1 where $(u, v) = (2, -3)$ and another 1 where $(u, v) = (-2, 3)$. Note: If you run the above code, you'll need to zoom in to see the holes.

Question 6
Using the code given for the 2-element interferometer example, generate the u, v-coverage plot in the rightmost panel of Fig. 7.5 and take its inverse Fourier transform. Display this with **imshow** and **mat2gray**, and sketch what you see. This is the response function of a 2-element interferometer.

```
M = fftshift(fft2(moon));
[Ly, Lx] = size(moon);
u = int32(-Lx/2 : Lx/2-1);
v = int32(-(Ly-1)/2 : (Ly-1)/2);
[Mu,Mv] = meshgrid(u,v);
bx = 2; by = -3;
uvplane = zeros(size(M));
uvplane(Mu == bx & Mv == by) = 1;
uvplane(Mu == -bx & Mv == -by) = 1;
rsp_function = ifft2(ifftshift(uvplane), 'symmetric');
imshow(mat2gray(rsp_function))
```



Question 7 (PHYS2911 and PHYS2921)
As a radio source (e.g. the Sun) drifts across the sky, constructive and destructive interference occur to produce a fringe pattern. The precision to which the position of such a radio source can be measured is determined by the fringe spacing, which sets the angular resolution of the interferometer. The closer the fringes, the higher the resolution. Based on what you know about Young's double-slit experiment, can you suggest two ways to improve the resolution of a 2-element interferometer?

In Young's double-slit experiment, the fringe spacing decreases as the slits move further apart, and also when a shorter wavelength is used. This means that you can improve the resolution of a 2-element interferometer by moving the receivers further apart, and/or observing at a higher radio frequency (shorter wavelength).

7.7 Activity: Simulating a multi-element interferometer

The effect of adding more receivers to an interferometric array is to increase the number of baselines. Each baseline adds a new pair of conjugate-frequency "holes" to the "aperture" of the interferometer, i.e. a new pair of spatial frequencies to the u, v-plane. The more complete the coverage of the u, v-plane, the more accurate the image of the sky that is produced. When designing a radio interferometer, people often perform simulations to see what the u, v-coverage will be like for different numbers and configurations of receivers. This allows them to optimise the layout of the array for particular science goals and to predict its response to different sky brightness distributions.

For this lab, you have been provided with a pre-written MATLAB function, `fillUVplane(x,y)`. This automates the processes of calculating the baseline vectors and u, v-coverage of an interferometer, given a list of positions of the receivers. It performs the same steps for simulating a 2-element interferometer that you saw earlier in Section 7.6, but now looped over all receiver pairs. `fillUVplane(x,y)` takes as input a list of 2D receiver coordinates, where the x and y-coordinates are stored in the variables `x` and `y` respectively, calculates the baselines, creates a 2D array of zeroes, and puts a 1 at the appropriate spatial frequencies. If you are curious, you can open up the function file to see how it works. The syntax for using this function is

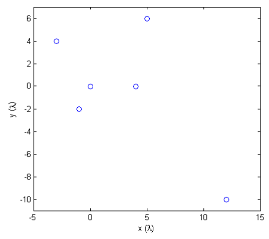
```
uv = fillUVplane(x, y);
```

where `uv` is the 2D array of ones and zeroes representing the "aperture" of the interferometer.

Question 8
We will now try out this function for a 6-element interferometer. This is the number of receivers that comprises the Australia Telescope Compact Array (ATCA), a world-class radio astronomical facility located in Narrabri, NSW about 500 km northwest of Sydney. The dishes of the ATCA are mounted on rail tracks and can be moved around into different configurations.

Let the ground coordinates of the six receivers be $\vec{r}_1 = (0, 0)$, $\vec{r}_2 = (12, -10)$, $\vec{r}_3 = (-1, -2)$, $\vec{r}_4 = (-3, 4)$, $\vec{r}_5 = (4, 0)$ and $\vec{r}_6 = (5, 6)$ in units of λ. Plot and sketch their layout. How many baselines does this interferometer have (hint: how many unordered pairs can you form from six elements)?

```
x = [0, 12, -1, -3, 4, 5];
y = [0, -10, -2, 4, 0, 6];
plot(x, y, 'o')
```



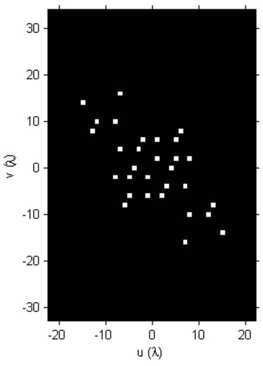
The number of unordered pairs you can form from n elements is ${}^{n}C_2 = n(n-1)/2$. A 6-element interferometer has ${}^{6}C_2 = 15$ baselines. Alternatively, the number of unordered pairs is equal to the sum from 1 to $n-1$, which equals $n(n-1)/2$.

Step 4: Find the image formed by the radio interferometer

Question 9

Use **`fillUVplane`** to compute and plot the u, v-coverage of this interferometer. Does the number of "holes" in this aperture match up with what you expect from your answer to Question 8?

```
u = int32(-Lx/2 : Lx/2-1);
v = int32(-(Ly-1)/2 : (Ly-1)/2);
uv = fillUVplane(x, y);
imshow(uv, 'XData', u, 'YData', v)
```

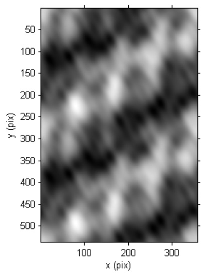


There are 30 "holes" in this aperture, which is consistent with what we expect from the number of baselines – there are 15 baselines, each contributing one pair of conjugate-frequency points in the u, v-plane ($15 \times 2 = 30$).

Question 10

Image the Moon using this interferometer, by convolving the response function with the Moon image. Recall that this involves multiplying the Fourier transform of the Moon image with the aperture (in this case, this is the array output by **`fillUVplane`**) and inverse Fourier transforming the result. Display the result using **`imshow`** and **`mat2gray`**, and describe what you see. Is this recognisable as the Moon?

```
moon6 = ifft2(ifftshift(M.*uv), 'symmetric');
imshow(mat2gray(moon6))
```



The image is full of blurry, wispy structure that bears little resemblance to the Moon.

What is going on? Well, from the u, v-coverage plot that you computed in Question 9, it is apparent that the u, v-coverage of this interferometer is quite sparse (only a small number of spatial frequencies are measured). The response function, which you can compute and plot using the commands

```
response = ifft2(ifftshift(uv), 'symmetric');
imshow(mat2gray(response))
```

is shown in Fig. 7.6. The complicated pattern of blobs you see in the response function are called "sidelobes". Sidelobes cause false structure to be imprinted on the image formed by an interferometer. This can be remedied by a process called *deconvolution*, which as the name suggests is intended to reverse the effects of convolution. However, this is beyond the scope of the material we will cover here.

The image you obtained for Question 10 looks the way it does because it is the Moon convolved with this messy-looking function. A raw image like that, before it has undergone deconvolution to remove sidelobe structure, is called a *dirty map*. No points for guessing why it is called this! In reality, interferometers with a small number of receivers like the ATCA can still make nice images by augmenting their u, v-coverage using a number of tricks thereby improving their response function, but we won't discuss those here.

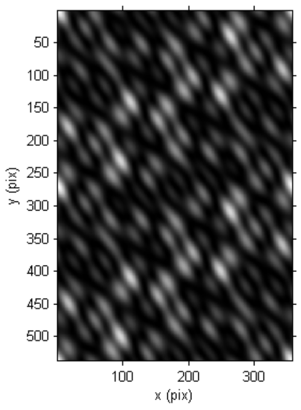


Figure 7.6: Response function of a 6-element interferometer

Question 11

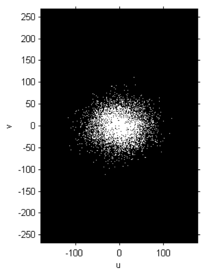
Let's now repeat this for a 128-element interferometer. This is the number of receivers that currently comprises the Murchison Widefield Array (MWA), a next-generation radio telescope in Western Australia that became operational in 2013. The MWA is a technology testbed for the future Square Kilometre Array, which will be the world's largest radio telescope and consist of thousands of receivers scattered across Australia and South Africa.

Generate a centrally-condensed distribution of 128 receivers using the commands

```
x = int32(round(20*randn(128,1)+Lx/2));
y = int32(round(20*randn(128,1)+Ly/2));
x(x < 1) = 1; x(x > Lx) = Lx;      % make sure x never falls outside of [1,Lx]
y(y < 1) = 1; y(y > Ly) = Ly;      % make sure y never falls outside of [1,Ly]
```

and then compute the u, v-coverage of this interferometer using `fillUVplane`. There are a large number of baselines, so `fillUVplane` may take a few seconds. `randn` is one of a number of inbuilt MATLAB functions for generating pseudo-random numbers (others include `randi`, which we will use in the Challenge section to this lab, and also `rand`). `randn` generates random Gaussian-distributed points.

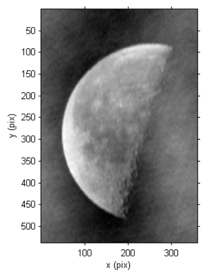
```
x = int32(round(20*randn(128,1)+Lx/2));
y = int32(round(20*randn(128,1)+Ly/2));
x(x < 1) = 1; x(x > Lx) = Lx;
y(y < 1) = 1; y(y > Ly) = Ly;
uv = fillUVplane(x, y);
imshow(uv, 'XData', u, 'YData', v)
```



Question 12

Image the Moon with the 128-element interferometer. How well can you see it now?

```
moon128 = ifft2(ifftshift(M.*uv), 'symmetric');
imshow(mat2gray(moon128))
```



There is still some wispy structure, but the Moon can now be seen clearly.

Checkpoint 2:

7.8 Challenge: Re-baselining the 128-element interferometer

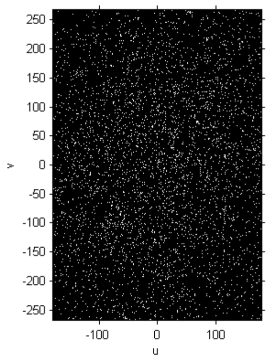
Let's now see what happens if we were to redistribute the 128 receivers of the interferometer over a much larger area. Instead of a centrally-condensed array, we are now going to spread them out in a uniformly random manner.

Question 13
Generate a uniformly-random distribution of 128 receivers using the commands

```
x = randi([1,Lx], 128, 1);
y = randi([1,Ly], 128, 1);
```

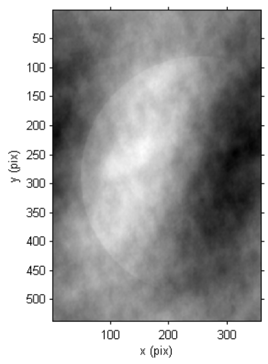
where **Lx** and **Ly** are the dimensions of the Moon image. Compute and display the *u*, *v*-coverage plot.

```
x = randi([1,Lx], 128, 1);
y = randi([1,Ly], 128, 1);
uv = fillUVplane(x, y);
imshow(uv, 'XData', u, 'YData', v)
```



Question 14
Image the Moon using this interferometer. How does it differ from the Moon image in Question 12?

```
moon128_unif = ifft2(ifftshift(M.*uv), 'symmetric');
imshow(mat2gray(moon128_unif))
```



There is more wispy structure in this image. It is possible to make out a sharp crescent outline, but the features/craters/etc. on the Moon are now difficult to see. The overall image of the Moon is not recovered as well as it was in Question 12.

Question 15
Can you explain why a centrally-condensed interferometer might be better for imaging a large object like the Moon? What about if you were trying to detect and measure the positions of small point-like objects, such as pulsars?

A centrally-condensed configuration is much more sensitive to smaller spatial frequencies, i.e. large-scale structure on the sky, than an interferometer spread out over a much larger area. The Moon is a large object and is therefore built mainly from small spatial frequencies, so it is imaged more accurately by a centrally-condensed array. On the other hand, interferometers with many long baselines are very sensitive to small-scale structure, as evidenced by the very sharp edge of the Moon when imaged with the more spread-out array. Point-like objects like pulsars would therefore be easier to detect with a spread-out array.