Fourier Transforms in Matlab

Matlab has a number of great functions implementing the FFT. Let's start with some simple examples:

```
>> x=[4 3 7 -9 1 0 0 0];
>> y=fft(x)

y =

6.0000
11.4853 - 2.7574i
-2.0000 -12.0000i
-5.4853 +11.2426i
18.0000
-5.4853 -11.2426i
-2.0000 +12.0000i
11.4853 + 2.7574i
```

The first element is real and is the zero-frequency part of the transform. In particular adding together the elements of x:

4+3+7-9+1+0+0+0 = 6.

Since the elements of x are spaced by ones, the sum is simply the integral of the input function or the area under the input function. Dividing by the number of samples yields the average of the input signal or it's "DC" value. The next three entries are the positive frequencies in the transform while 18.0 is the transform value at the Nyquist frequency. $f_N=1/(2\Delta t) = 0.5$ Hz (given $\Delta t=1$). The final three elements are the negative frequency values. If you compare the negative and positive frequency values, you see that the amplitudes are the same, but the actual values differ in the sign of the imaginary value. $y(\omega)=y(-\omega)^*$ where '*' indicates the complex conjugate. That is, the transform is Hermitian as indicated earlier in the notes. The Fourier Transform of a real function is Hermitian. Let's see how this can be packed in a more rational way:

```
z=fftshift(y)

z =

18.0000
-5.4853 -11.2426i
-2.0000 +12.0000i
11.4853 + 2.7574i
6.0000
11.4853 - 2.7574i
-2.0000 -12.0000i
-5.4853 +11.2426i
```

Now the DC value is in the center with the negative frequencies lying above the Nyquist until the DC value is reached. The remaining frequencies are positive with the complex conjugate values of the negative frequencies.

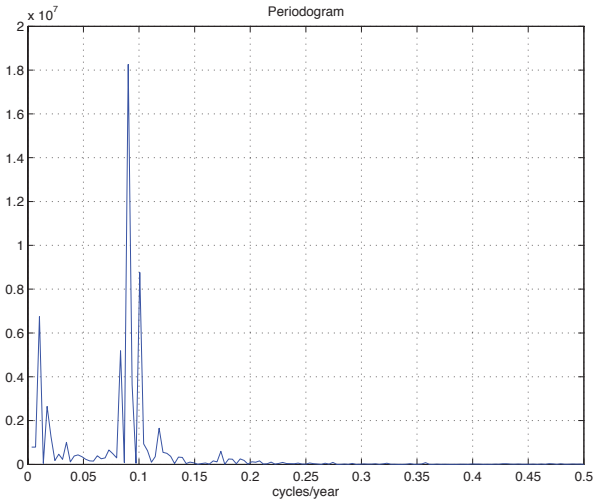

Figure 1: Sunspot time series.

Fourier Transforms are helpful not only in solving partial differential equations, but can be very helpful in data analysis. A good example, with a data table built into Matlab itself, has to do with the variations in sunspot activity during the past 300 years. You likely know that sunspot activity occurs in cycles with a period of approximately 11 years. It's a fairly significant outcome of the processes that drive the magnetohydrodynamics of the Sun. Astronomers have tabulated a *Wolfer* number that is representative of both the number and size of sunspots.

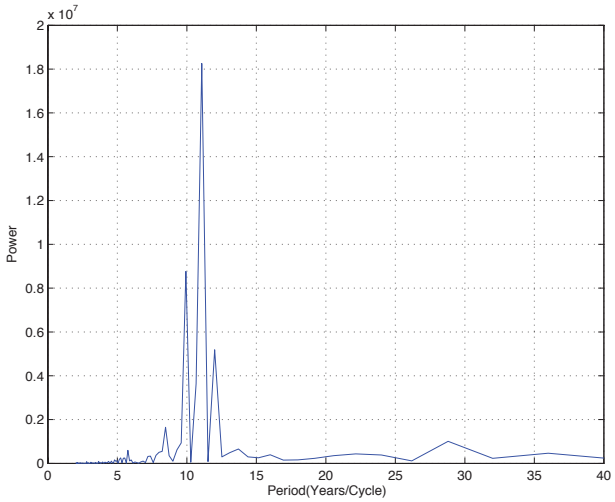

Figure 2: Power spectrum for sunspots

```
>> load sunspot.dat
>> year=sunspot(:,1);
>> wolfer=sunspot(:,2);
>> plot(year,wolfer)
>> title('Sunspot Data')
>> xlabel('Year')
>> ylabel('Wolfer Number')
```

Matlab loads a dataset installed with Matlab called "sunspot." The data in sunspot are arranged in two columns with year in the first column and the Wolfer number in the second. The time series are shown in Figure 1. Now take the FFT of the sunspot data.

```
>> Y=fft(wolfer);
>> N=length(Y);
>> Y(1)=[];
>> power=abs(Y(1:N/2)).^2;
>> nyquist=1/2;
>> freq=(1:N/2)/(N/2)*nyquist;
>> plot(freq,power), grid on
>> xlabel('cycles/year')
>> title('Periodogram')
```

The result is a complex vector, Y. The magnitude of Y squared is called the power and a plot of power versus frequency is a periodogram. The first component of Y is removed (the *DC* component). The periodogram is shown in Figure 1. A peak is readily seen in the signal at a frequency of 1/11 years. This would probably be more obvious if the function was plotted in years/cycle (the inverse of the x-axis in Figure 1).

```
>> period=1./freq;
>> plot(period,power), axis([0 40 0 2e7]), grid on
>> ylabel('Power')
>> xlabel('Period(Years/Cycle)')
```

The peak at 11 years is now quite obvious (Fig. 2). If we look for the peak in the power spectrum, we can determine the sunspot period more precisely:

```
>> [mp index] =max(power);
>> period(index)
```

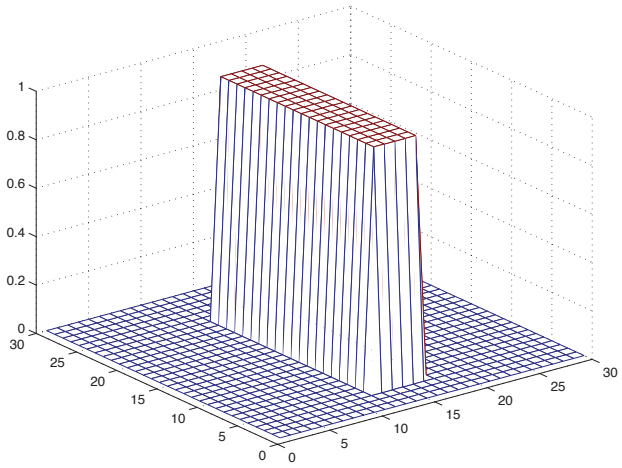

Figure 3: Gate function in 2-D.

ans =

11.0769

So the period, based on 300 years of data, is slightly greater than 11 years! The utility of the FFT in data analysis should be pretty obvious. An entire course can be devoted to *time series analysis* and the use of the FFT!

Just a quick note, don't miss the subtle use of .^ and ./ above to denote element-by-element operations on vectors.

2-D Fourier Transforms

Let's have a look at some two-dimensional Fourier Transforms now.

```
>> [x,y]=meshgrid(1:30);
>> Z=zeros(30,30);
>> Z(5:24,13:17)=1;
>> mesh(x,y,Z)
>> [x,y]=meshgrid(1:30);
```

This creates a 2-D gate function or box in Matlab with different horizontal dimensions in the x,y directions with a value of 1 within the box. The function is plotted in Figure 3. The Matlab functions fft, fft2 and fftn implement the Fast Fourier Transform for computing the 1-D, 2-D and N-dimensional transforms respectively. The inverse functions ifft, ifft2 and ifftn compute the inverse transforms.

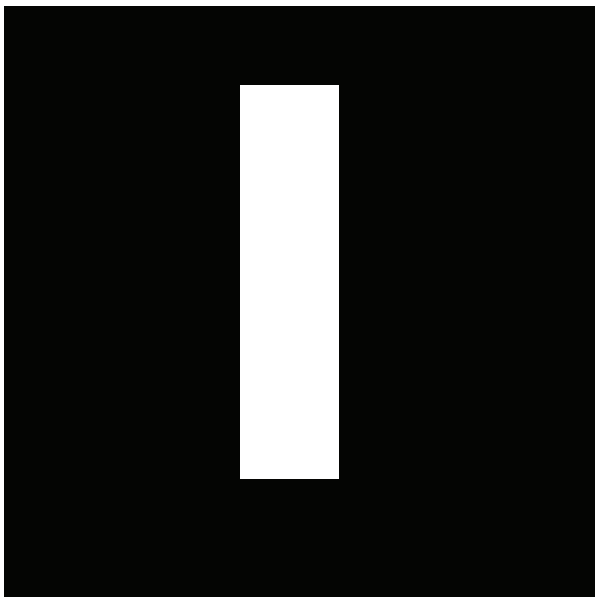

Figure 4: Top view of gate function.

Note that the function is long in the y-direction and short in the x-direction. This will affect the Fourier

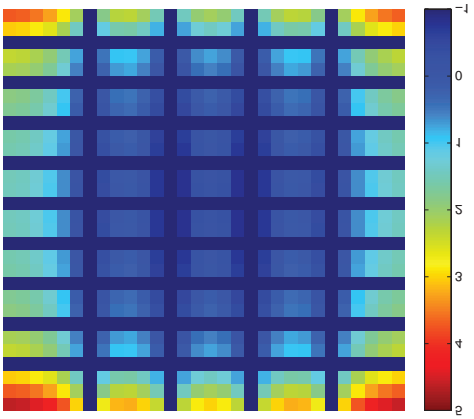

Figure 5: Fourier transform at low resolution.

transform views in the linerarly-related frequency or wavenumber domain. We'll plot this function first with another Matlab routine called "imshow:"

>>imshow(Z,'InitialMagnification','fit')

This provides an alternative view of the box with amplitude = 1 at the center of the 2-D field (Fig.4). Now let's take the Fourier transform:

>>F=fft2(Z);
>>F2=log(abs(F));
>>imshow(F2,[-1 5],'InitialMagnification','fit');
colormap(jet); colorbar

The Fourier Transform (Fig. 5) just computed is quite low resolution and the zero component (or DC) is still in the upper left corner instead of at the center. Let's pad the Fourier Transform to 256 points in each direction and have another look.

>>F = fft2(W,256,256);

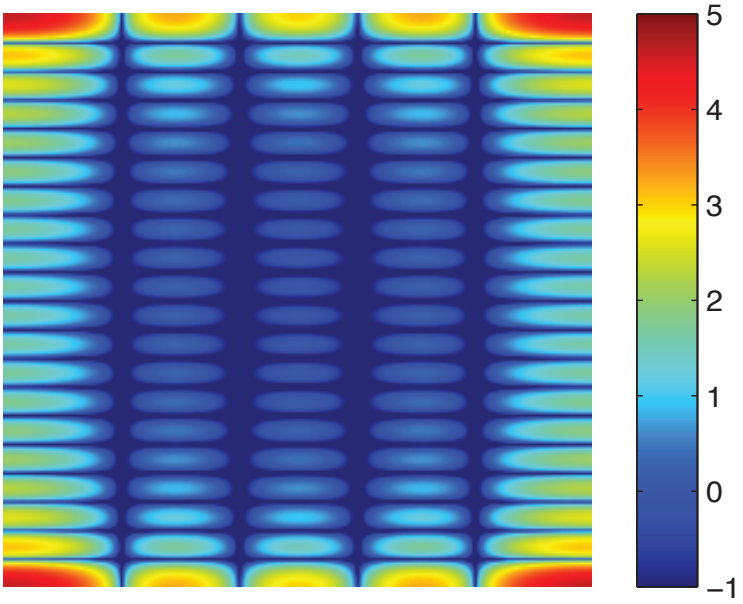

Figure 6: Higher resolution Fourier Transform. The DC component is still inappropriately located in the plot.

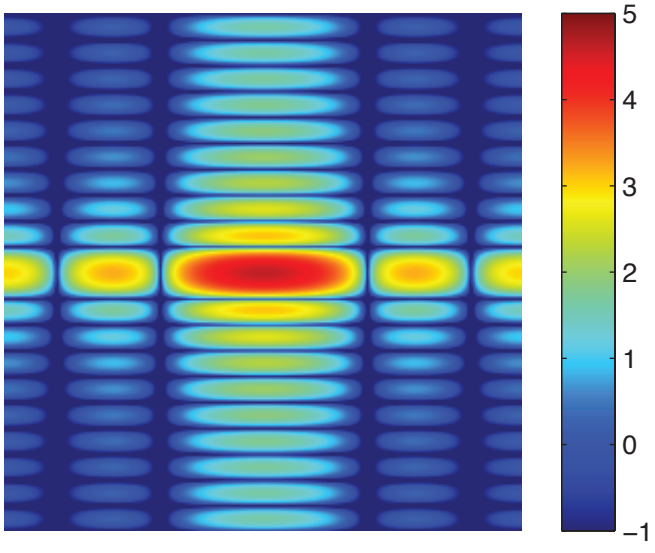

Figure 7: High resolution 2-D Fourier Transform with the DC component located in the center of the plot. Recall that the original gate had a long dimension in the y-direction and a short, localized dimension in the x-direction. The frequency content in the x-direction decays less rapidly than in the y-direction. A localized object in the x or t direction requires a broad spectrum and vice-versa.

>>imshow(log(abs(F)),[-1 5]); colormap(jet); colorbar

This version (Fig 6) has a much higher resolution although we have not yet moved the zero frequency amplitude to the center. Let's do that.

>>F2 = fftshift(F)
>>imshow(log(abs(F2)),[-1 5]); colormap(jet); colorbar

Figure 7 now shows the 2-D spectrum in high reslution with the DC component in teh correct place.

It's possible to view the 2-D spectrum differently. Figure 8 shows a mesh presentation of the Fourier Transform of the 2-D gate function.

>>mesh(log(abs(F2))

Recall from the lecture notes that a 1-D gate function transforms into a sinc function of $\sin(\pi x)/(\pi x)$.

Let's look at a simple 1-D example. First create a gate function:

>> t=zeros(256);
>> t(122:132)=1;

Now take the Fourier Transform:

>> w=fft(t);

```
>> x=fftshift(w);
```

This gives us the properly arranged FFT. Now generate a gate function that is broader:

```
>> t(102:152)=1;
>> w1=fft(t);
>> x1=fftshift(w1);
>> plot(x,'r',x1,'b')
```

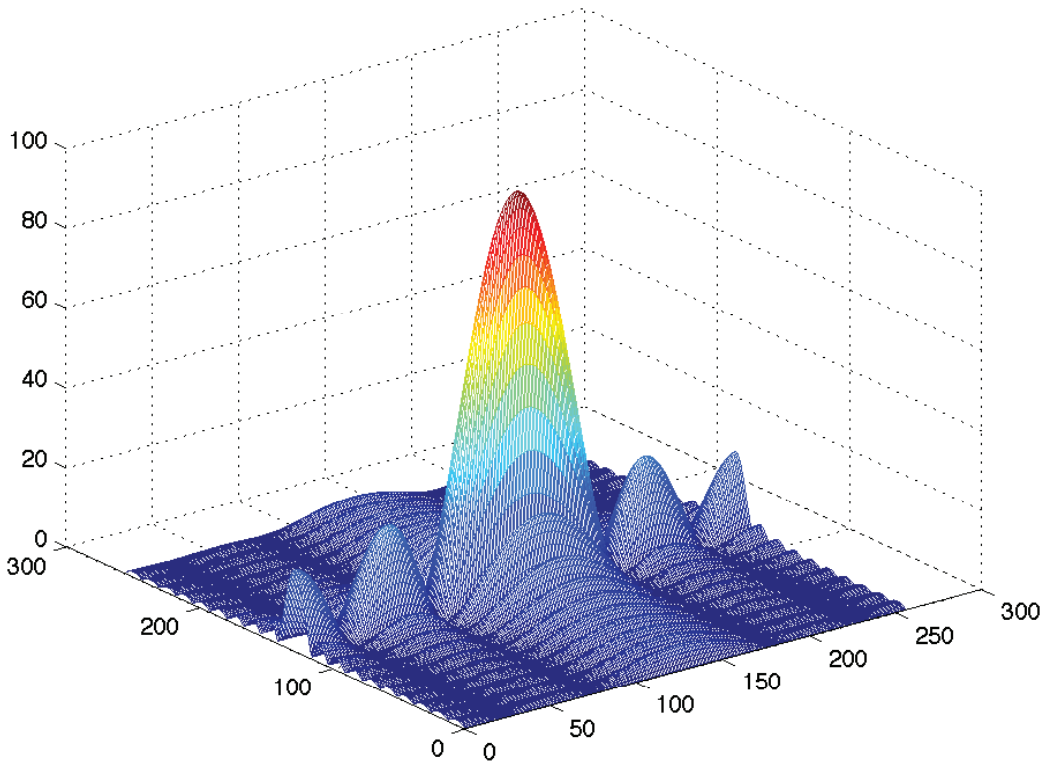

Figure 8: Mesh plot of Figure 7. Now it's possible to see clearly the sinc functions in both the x & y directions rather than simply color graphics. All of these figures are itegrated into the notes in encapsulated postcript format (eps) so the rendering on the page is a bit spotty. Looks best when printed on a good color printer.

Now plot the narrow gate function in red followed by the broader function in blue:

```
>> plot(abs(x),'r')
>> hold on
>> plot(abs(x1),'b')
```

Figure 9: Two Fourier Transforms of gate functions. The blue function correspondes to the broader gate function resulting in a more localized sinc function in the frequency domain. This is the same effect as noted in Figures 5-8 above.