Taylor Rembos
EEL 3135
Lab 8 Part 1

Lab 9: An Introduction to Using the FFT Function in MATLAB

1.1 Definition: What is the FFT?

FFT is an acronym for the *Fast Fourier Transform*, an algorithm that computes the discrete Fourier transform (DFT) of a discrete-time signal. In other words, the FFT takes a *discrete-time* signal and extracts important *frequency domain* information from that signal. It is considered "fast" because its computational complexity is far less than that of the DFT, the Discrete Fourier Transform. In this lab, we won't care about the mechanics of the algorithm (hurrah!); we are simply concerned with how to use the FFT function in MATLAB to determine frequency domain information of signals.

1.2 Background: How does the FFT "Read" Frequency Information?

Before starting the lab, it may be useful to gain some insight as to how the discrete Fourier transform extracts frequency information from a discrete input signal. The FFT utilizes a series o f *sinusoidal basis signals* of varying frequency. The FFT determines how closely the input signal "matches" these basis signals, and outputs a vector of complex numbers - one for each iteration of the algorithm. In this lab, we will call these complex measurements *frequency bins*, or simply *bins.*

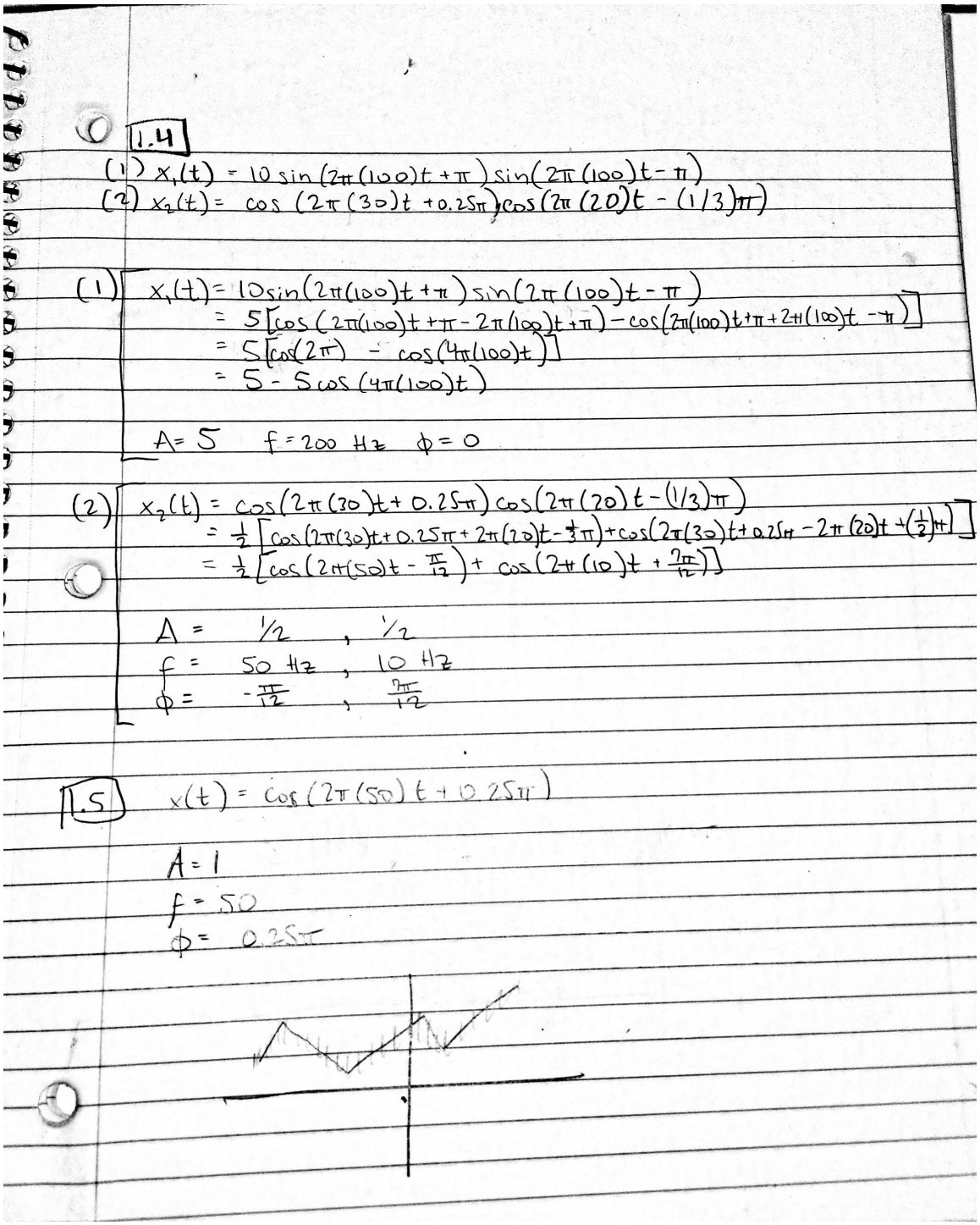
Because the outputs of the FFT are complex, we need an easier way to visualize them. Simply plotting the vector of frequency bins will result in a complicated jumble of vectors in the complex plane. Therefore, we "read" the results of the FFT by plotting the *magnitude spectrum* of the frequency bins. The higher the magnitude of the frequency bin, the higher the *correlation* between the input signal and a sinusoid of that frequency. Therefore, the magnitude spectrum should spike at regions where the frequency of the sinusoidal basis signals match the input signal. Recall that any signal can be expressed as the sum of sinusoids of different frequencies. Consequently, the FFT is an extremely powerful tool because it can help us decompose a complicated discrete-time signal into its sinusoidal components.

1.3 Useful Lecture Series:

For more information on the DFT and MATLAB FFT function, refer to the following series of lecture videos by David Dorran. They are very useful for gaining an intuitive understanding of the DFT and FFT, and will prove useful in completing this lab.

1.4 Exercise: Decomposing Signals into Sinusoids

Rewrite the following signals, by hand, as a sum of sinusoids (that is, expand them as much as possible). Identify the amplitude, frequency, and phase of each sinusoidal component. In your write-up, provide your answers with explanations or scans of your work. (*Note:* this should be a trivial task that serves essentially as a review of the beginning of the course)

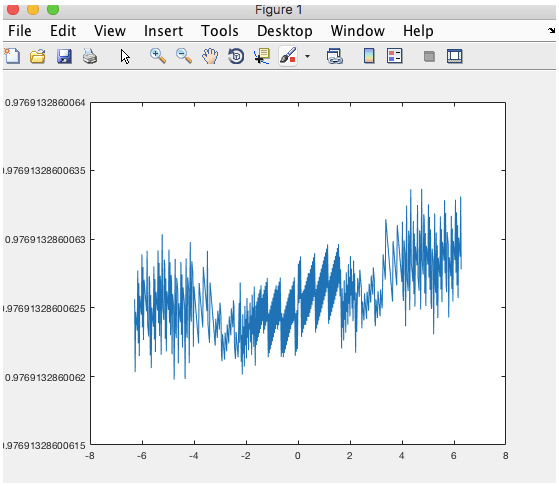


1.5 FFT of a Single Sinusoid with Integer Frequency

$$x(t) = \cos(2\pi(50)t + 0.25\pi)$$

Plot the continuous-time frequency spectrum of this signal by hand.

```
>> fs = 50;
>> A = 1;
>> phi = 0.25*pi;
>> t = -2*pi:(1/fs):2*pi;
>> x = cos(2*pi*50*t + 0.25*pi);
>> plot(t,x)
```



1) Define the variable fs for the sampling frequency, and assign it 1000 samples/second.

2) Construct a time vector using this sampling frequency that is exactly 1500 samples long, *starting at zero, not at 1/fs*.

3) Define $x(t)$, the sinusoid described above, as the vector xx.

4) Run the FFT function on xx. The function fft() only takes one argument: the input signalxx. We will use a semicolon to suppress the output (otherwise the command window will fill with complex numbers). By convention, frequency domain variables are denoted with capital letters, so we will use XX to store the outputs.

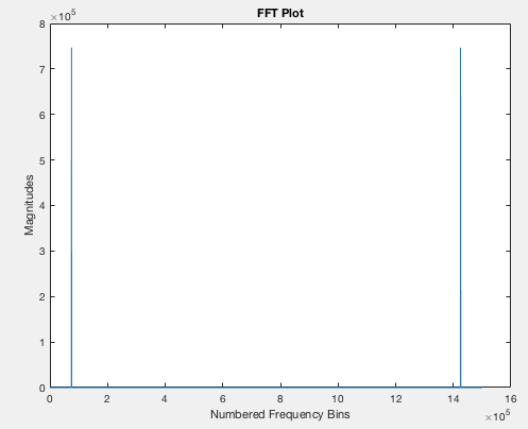
>> XX = fft(xx);

```
>> fs = 1000;
>> t = 0:(1/fs):1500;
>> xx = cos(2*pi*50*t + 0.25*pi);
>> XX = fft(xx);
```

a. Create a new vector that stores the values of abs(XX) and plot it with the `plot(abs(XX))` command. This plot is called the *magnitude spectrum*, where the x-values are numbered frequency bins and the y-values are their corresponding magnitudes.

b. Compare this plot to your continuous-time spectrum. What's different about it?

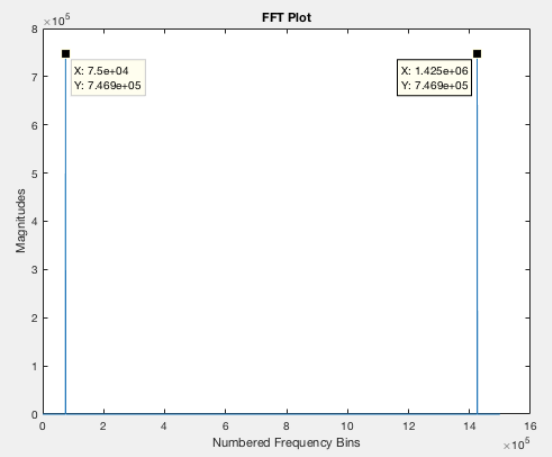
```
>> v = abs(XX);
>> plot(abs(XX))
>> xlabel('Numbered Frequency Bins')
>> ylabel('Magnitudes')
>> title('FFT Plot')
```



This plot is the opposite of my continuous-time spectrum plot with the minimum and maximum x values being the largest and equivalent and all of the in-between values being around 0.

c. In the plot of the magnitude spectrum, you should notice two congruent peaks on either side. For the signals we will be dealing with in this lab (real signals), the magnitude spectrum will be two sided. That is, any peak will be a mirror image of the peak on the other side. Because of this redundancy, we can simply evaluate the peak on the left.
What is the vector index corresponding to the left peak? Hint: use the find function
What is the value of the magnitude spectrum everywhere else?

```
>> I = find(7.5e+04)

I =

     1

>> I2 = find(1.425e+06)

I2 =

     1

>> [I, J, V] = find(7.5e+04,7.469e+05)
```



I =

1

J =

1

V =

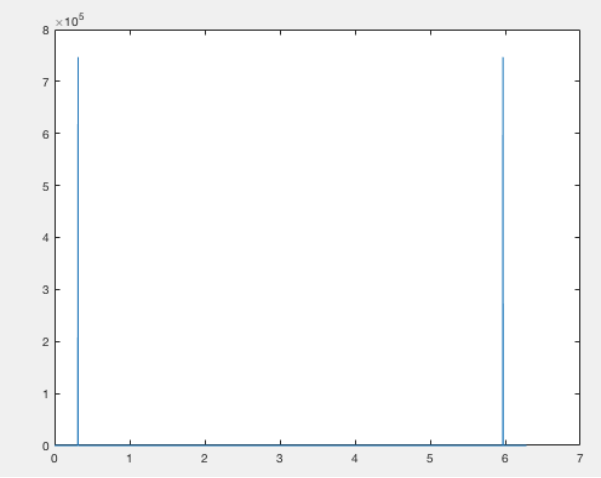
75000

The value everywhere else is 0.

d. The result of the Fourier transform of a sampled signal goes into frequency bins that correspond to the normalized radian frequency, ˆω. Our convention thus far has been to draw magnitude spectra from –pi to pi. MATLAB's FFT function, however, returns frequency bins ranging from 0 to 2pi. Plot the MATLAB FFT output against normalized radian frequency. Here's the command for the normalized radian frequency output:

ww = 0:(2*pi/length(XX)):(2*pi-1/length(XX));
plot(ww,abs(XX));

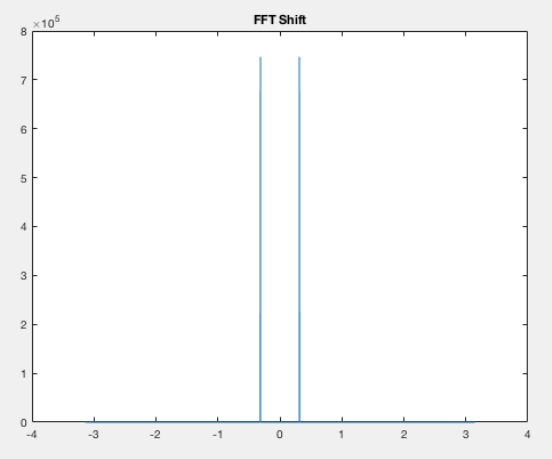
```
>> ww = 0:(2*pi/length(XX)):(2*pi-1/length(XX));
>> plot(ww,abs(XX));
```



e. MATLAB's fftshift function is a way to "fix" the output of the MATLAB FFT function in order to make it range from –pi to pi. Plot the shifted MATLAB FFT output against normalized radian frequency:

ww = -pi:(2*pi/length(XX)):(pi-1/length(XX));
plot(ww,abs(fftshift(XX)));

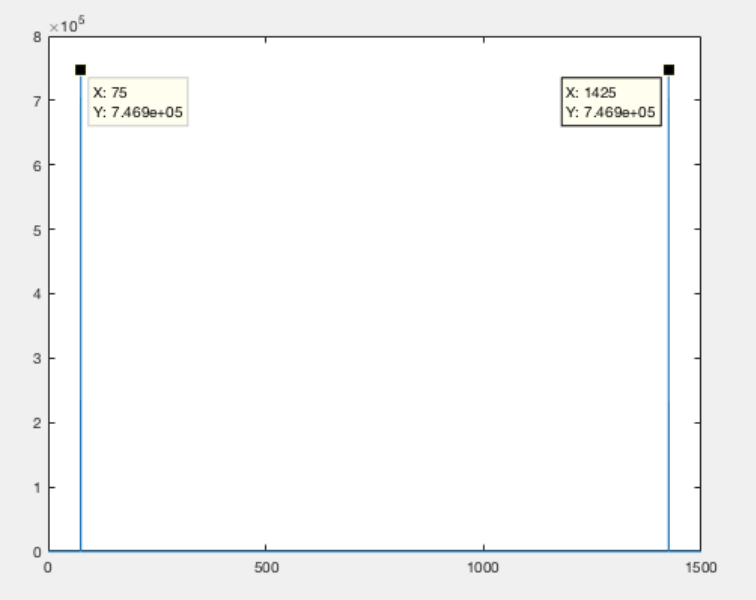
```
>> ww = -pi:(2*pi/length(XX)):(pi-1/length(XX));
>> plot(ww,abs(fftshift(XX)));
>> title('FFT Shift')
```



f. The normalized radian frequency is related to the frequency in Hz by the sampling frequency and a factor of 2pi. Plot the FFT of XX against the Hertz frequencies of the bins. What are the frequencies matching the peaks?

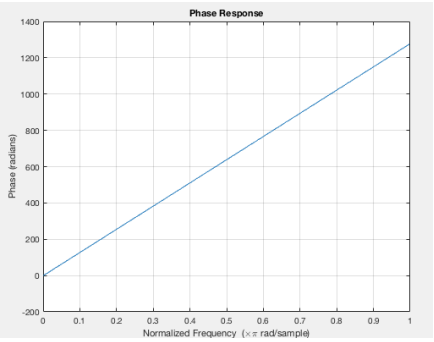
```
>> plot(t,abs(XX));
```

The frequencies matching the peaks are 750000 at x = 75 and 750000 at X = 1425.



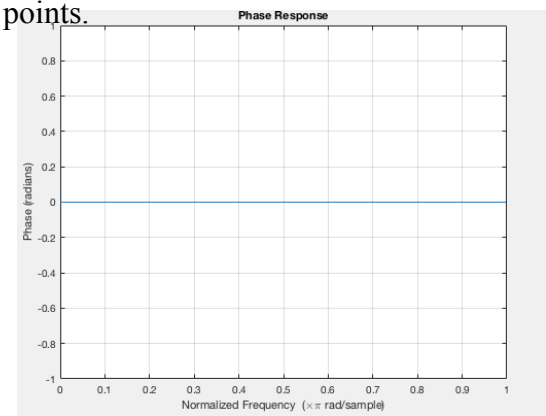
6) The FFT can also be used to extract *phase* information from the signal. From the complex amplitude of the MATLAB FFT function, you can use the angle() or phase() function to find the phase in each bin. Plot the phase spectrum of the signal against Hertz frequency. What are the phases of the peaks? Does this match what you would expect, and why?

```
>> phasez(XX)
```



>> phasez(75)
>> phasez(1425)

Phase at both 75 and 1425 is 0. This matches what we would expect because at this point, xx is simply cos(2*pi*50*t) and does not include a phase angle, which would result in the phase being 0 at these maximum points.

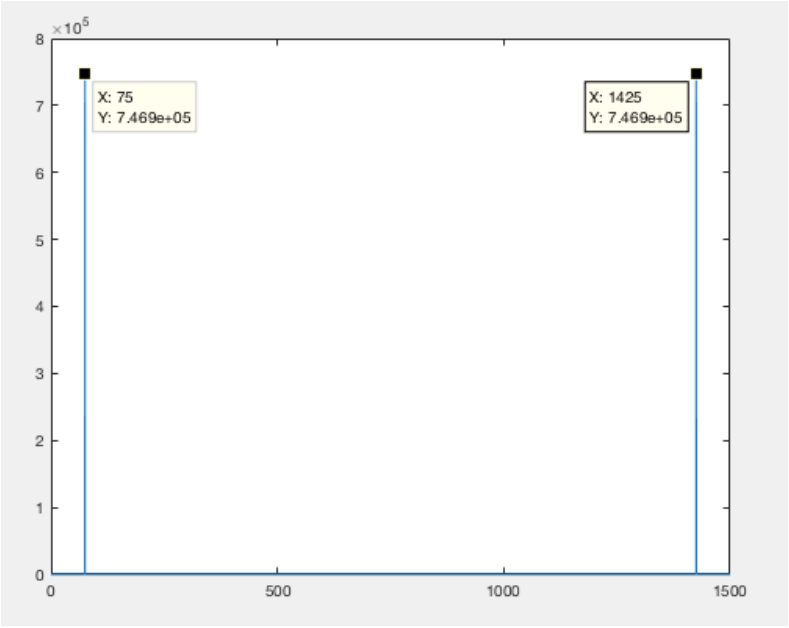


7) In parts (5) and (6), we identified the frequency and phase of $x(t)$. It turns out that we can also use the FFT to determine the exact magnitude of $x(t)$, by finding the value of the magnitude spectrum in that frequency bin. Determine the exact magnitude of the peak. Note that the two peaks will have the same magnitude, being complex conjugates of each other.

From parts (5) and (6) we can see:

>> plot(t,abs(XX));

The magnitude of the peak is 746000.



1.6 FFT of Multiple Sinusoids with Integer Frequency

1) Using the same sampling frequency and duration from part 1.5, create the vectors x1 andx2, storing 1500 samples of the signals $x_1(t)$ and $x_2(t)$.

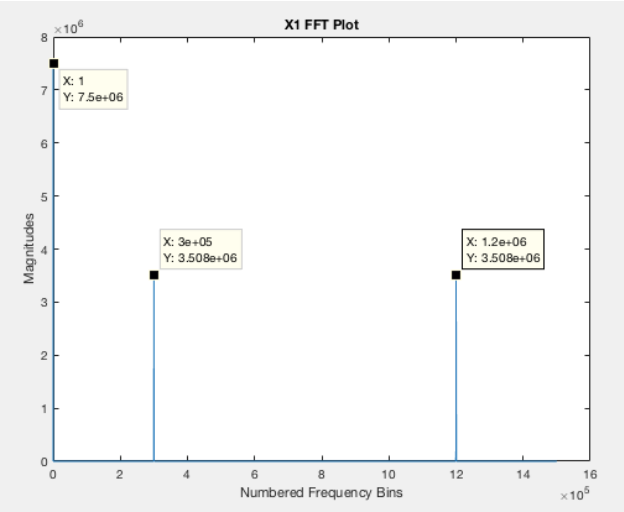
2) Plot the *magnitude spectra* o f x1 and x2 against Hertzian frequency. Unlike in 1.5, we should observe multiple peaks, each corresponding to a single sinusoidal component. Comment on the appearance of each spectrum. Do you notice any differences in the number of peaks for these signals?

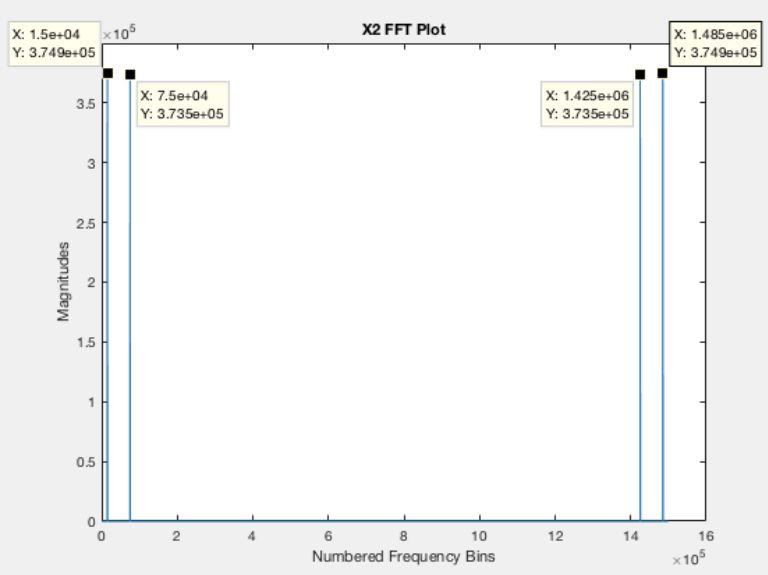
3) Using MATLAB, find the frequencies, amplitudes, and phases of the sinusoidal components of x1 and x2. Compare these results to what you found by hand.

f = 50 Hz
phase = pi/4 radians
fs = 1000 samples / sec
sampling frequency 1500 samples long starting at zero

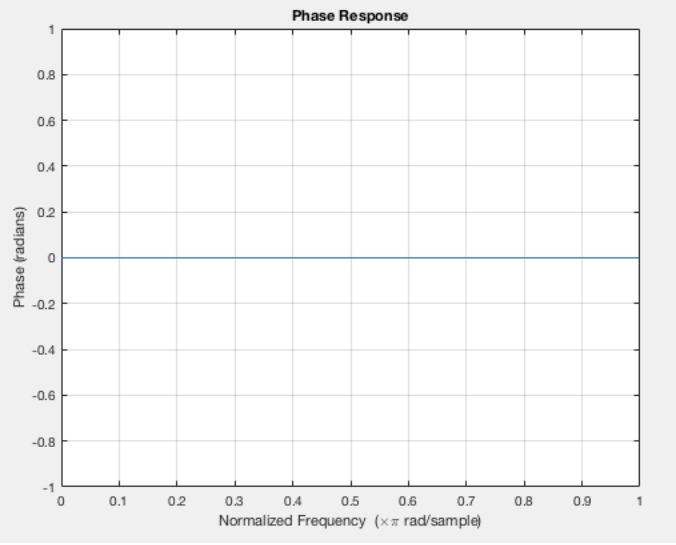
```
>> f = 50;
>> phase = pi/4;
>> fs = 1000;
>> t = 0:(1/fs):1500;
>> x1 = 5 - 5*cos(4*pi*100*t);
>> x2 = 1/2*[cos(2*pi*50*t - (pi/12)) + cos(2*pi*10*t + (7*pi/12))];
>> XX1 = fft(x1);
>> plot(abs(XX1));
>> title('X1 FFT Plot')
>> xlabel('Numbered Frequency Bins')
>> ylabel('Magnitudes')
>> XX2 = fft(x2);
>> plot(abs(XX2));
>> title('X2 FFT Plot')
>> xlabel('Numbered Frequency Bins')
>> ylabel('Magnitudes')
```

The X1 plot has 3 peaks, the maximum being the lowest point on the x scale and a mirror image of the smaller peaks in the middle.
The X2 plot has 4 peaks, mirror images of the same magnitude, with the smaller magnitudes being the peaks toward the middle and the larger being the outside peaks.





The frequencies and amplitudes of the plots are similar but not exactly the same.
For X1 plot, the phases are all 0 as calculated.



>> phasez(1)
>> phasez(3e+05)
>> phasez(1.2e+06)

For X2 plot, the phases are all also 0,
which is off by my calculations of pi/12 and –7*pi/12.
>> phasez(7.5e+04)
>> phasez(1.425e+06)
>> phasez(1.485e+06)
>> phasez(1.5e+04)