Here’s some background info for you, since today’s material goes beyond what is covered in the book. Make sure you read and understand this - it defines the conventions we will use in this class.

When we evaluate real signals in a lab, the computer (oscilloscope, graduate student, etc.) only takes data at discrete times. So rather than a continuous function from \( t = -\infty \) to \( \infty \), we have a finite list of numbers taken at specific times. We can still write our data as a sum of sines and cosines or complex exponentials, and that sum will be an exact representation of our data. But because we have discrete points, and don’t have a measure of the signal at every time, this sum will not be a perfect representation of the signal. And the amplitudes of the sines and cosines may or may not give us a good idea of what frequencies are present in our signal. To see why, look at the images below.

In the upper left hand image, we see a sine wave (the black line) which is being sampled at a fairly fast rate (the round circles represent data points). If we connect the data points, we have something that oscillates at the frequency of the sine wave. Moving to the upper right-hand image, the sample rate slows down to twice the frequency of the sine wave. This is just fast enough to resolve the sine wave. If we sample any slower - as seen in the lower left-hand image, we get data that appears to be oscillating - but at the wrong frequency! This is known as “aliasing.” Our data makes it look like a frequency is there which isn’t. If we sample at exactly the frequency of the sine wave (the lower right-hand image), we get something that looks like a zero-frequency wave.

So what did we learn from this?

• First, we learned that if the thing we are measuring has frequency components higher than half the sample rate, we will get “artifacts” due to “aliasing” when we do our transform - we will see power appear at lower frequencies when the power is really at higher frequencies. So if you are doing your Fourier transform to try to see what frequencies are in your signal, you could be fooled.

• Second, we learned that if I take data points spaced by a time interval \( \tau \) the maximum frequency component that we can extract from our data is a frequency of \( v_{\text{max}} = 1/2\tau \). This maximum frequency is known as the “Nyquist frequency.” If my signal has frequencies higher than the Nyquist frequency, they will be aliased down to lower frequencies. Furthermore, when I write my data as a sum of sines and cosines or complex exponentials, there will be a finite range of frequencies in my sum.

The next problem that we have with “real data” is that we only take data for a finite amount of time \( T \). What happens before or after that? Our data doesn’t tell us, so there’s no answer. But as far as doing a Fourier transform, it’s convenient to pretend that it repeats itself over and over again. Remember when we did Fourier series of periodic signals? We only had discrete frequencies in our sum. As such, when we perform a discrete Fourier transform, not only will it have a finite range of frequencies in it, it will only have discrete frequencies within that span. So our sums will have a finite number of terms.

Despite these limitations, there are “fixes” which allow us to pull a spectrum out of a finite set of discrete data points. But while the spectrum we get is useful, we have to remember that it isn’t an “exact” representation of the thing that we measured. The Fourier transform will, however, be an exact representation of our data. The problem isn’t with the transform, it’s with our data! Since our data doesn’t have complete information about the signal we measured, our Fourier transform also won’t have complete information about the signal. But it WILL contain all of the information in our data set.
If I have $N$ data points, each taken a time $\tau$ after the previous point, and if I label each data point as $f_k$ where $k = 0, 1, 2, ..., N - 1$, we can express that data set in the form

$$f_k = \frac{1}{\sqrt{N}} \sum_{n=-N/2}^{N/2} c_n e^{\frac{2\pi i n k}{N}}.$$ 

Or, we can recognize that $k = t/\tau$, and that $\omega_0 = 2\pi/T = 2\pi/N\tau \Rightarrow N = 2\pi/\omega_0 \tau$, and write this as

$$f(t) = \frac{1}{\sqrt{N}} \sum_{n=-N/2}^{N/2} c_n e^{i\omega_0 t}.$$ 

Note the sign of the exponent - it is different from your textbook. This is the more common convention and is consistent with the convention used in the book for regular Fourier transforms. This is the convention we will use in this class. Note that the highest frequency term, $c_{N/2}$, corresponds to an angular frequency of $(N/2)\omega_0 = \pi/\tau$, or a frequency in Hz of $1/2\tau$ - the Nyquist frequency. The lowest frequency term, $c_{-N/2}$, corresponds to a frequency of $-1/2\tau$. The terms are spaced in angular frequency by an amount $\omega_0 = 2\pi/T$. That means, in Hz, they are spaced by a frequency which is just one over the total sample time.

You may be wondering why the Nyquist frequency is half the sampling frequency rather than the sampling frequency. Hopefully, looking at the plots above, you can figure out why. According to Nyquist’s theorem, if you want to detect a signal at a frequency $\nu$, you had better take samples at a frequency at least twice as fast as $\nu$.

The expression that tells us what the $c_n$ coefficients are is

$$c_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} f_k e^{-\frac{2\pi i n k}{N}}.$$ 

This process is known as a “discrete Fourier transform,” or DFT. The previous formula which gives us $f$ as a sum involving the $c_n$ coefficients is the inverse discrete Fourier transform or IDFT. If we calculate the $c_n$ coefficients, and then do an inverse Fourier transform, we will get our data points back. But if we plug a time into the inverse transform that doesn’t correspond to the time we took one of our data points, the formula will interpolate for us.

1. (13 points) Use some mathematical package on the computer to do the following. (Hint, save your code, because you’ll reuse it in the next problems.)

   (a) Define an array of numbers $f_k = \sin(16\pi k/N)$ where $N = 100$ and $k = 0, 1, 2, 3, ..., 99$. Plot $f_k$ and attach the plot to your homework. (Hint - if you do this in Mathematica or Maple, make sure your code evaluates the sine to a number - i.e. N[\sin[16*Pi*k/NN]] rather than \sin[16*Pi*k/NN])

   (b) Since you have a pure sine wave with an amplitude of 1, when you perform the DFT you should get a bunch of $c_n$ terms that are zero, and only the ones corresponding to the correct frequency will be non-zero. For which values of $n$ will $c_n$ be non-zero? (Hint - you don’t actually know the frequency of the sine wave, because I didn’t tell you what $\tau$ is. Just pretend like you know $\tau$ - it will go away in the end).

   (c) Have the computer calculate the array of numbers $c_n$, and then plot $|c_n|$ as a function of $n$. Attach a printout of your code and the plot to your homework.

   (d) Because of round-off errors, your $c_n$ values that should have been zero will probably not be quite equal to zero - but they should be very small except for two of them. For which values of $n$ is $c_n$ not extremely small? Are the values of $n$ you predicted?

   (e) Now do the inverse transform, and plot the real and imaginary parts of the resulting $f_k$ array. Attach a printout of your code and the plot.

2. (7 points) Now let $f_k = \sin(14.7\pi k/N)$.

   (a) Plot $f_k$.

   (b) Plot $f_k$ and $|c_n|$ vs $n$. Notice that this time, there is a spread of $c_n$’s that are non-zero. Why?

   (c) Do the inverse transform and plot the real and imaginary parts of the resulting $f_k$ array.

3. (6 points) Now let’s see what happens when we get frequencies close to half the sampling rate.

   (a) Let $f_k = \sin(98\pi k/N)$. Plot $f_k$ and $|c_n|$. Note that even though $f_k$ doesn’t look much like a sine wave, $c_n$ is still smart enough to find the sine wave and give you the right frequency.
(b) Now let \( f_k = \sin(190\pi k/N) \). Plot \( f_k \) and \(|c_n|\). Note that we get the spikes in \(|c_n|\) at the wrong frequency! This is aliasing, my friend. Also note, however, that the inverse transform is (within round-off error) the same as \( f_k \). The problem is NOT with the transform, but with the data you started with.

4. (6 points) Now let’s see how Fourier transforms can pull a signal out of a mountain of noise. Let \( f_k = 0.2 \sin(30\pi k/N) + 0.3 \sin(14\pi k/N) + b_k \) where \( b_k \) is just a string of random numbers such that each \( b_k \) has a random value between -0.5 and 0.5.

(a) Plot \( f_k \).

(b) Calculate and plot \(|c_n|\).

5. (8 points) Imagine that I took the discrete Fourier transform of a set of data and got a set of coefficients \( c_n \). But now I want to think of my data as being made up of sines and cosines:

\[
f_k = a_0 + \sum_{n=1}^{N/2} a_n \cos \left( \frac{2\pi N}{N} n k \right) + b_n \sin \left( \frac{2\pi N}{N} n k \right).
\]

Use Euler’s formula to find a formula for \( a_0 \), \( a_n \), and \( b_n \) in terms of \( c_n \). Note that there is no factor of \( 1/\sqrt{N} \) in the expression above, so you should expect there to be a \( \sqrt{N} \) in each of your formulas. Note that you can check your formulas by taking the DFT of a sine or cosine and seeing if you get the correct amplitudes for the sine or cosine.