1. (6 points) Because complex exponentials are related to sines and cosines, it turns out that you can represent a periodic function as a sum of complex exponentials. But unlike sines, the sum will include \textit{negative} frequencies:

\[ f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega_0 x}. \]

The reason for this will hopefully be clear as we work this problem.

(a) Start with a standard Fourier series:

\[ f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(nk_0 x) + b_n \sin(nk_0 x). \]

Now insert the identities below that we found earlier in the semester and show that you can write the function as a sum of complex exponentials of the form given above, and find equations that give \( c_n \) in terms of \( a_n \) and \( b_n \).

\[ \cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}, \quad \sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i} \]

(b) Use these identities to find the integral formula you would use to find the \( c_n \) terms when writing a periodic function as an exponential Fourier series. In other words, find the Euler formula for \( c_n \) from the Euler formulas for \( a_n \) and \( b_n \). Make sure your final answer is an integral involving a complex exponential, not involving sines or cosines.

2. (8 points) Consider \( f(t) \), pictured below. This wave is equal to 0 from \( t = 0 \) to \( t = 0.5 \). Then from \( t = 0.5 \) to \( t = 1 \) it ramps linearly from 0 to 1. Then it repeats.

(a) Write this wave in the form

\[ f(t) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega_0 t}. \]

Note that

\[ \int u e^{-iu}du = (1 + iu)e^{-iu} + C \]

and don’t let the fact that we are using \( \omega_0 t \) rather than \( k_0 x \) confuse you - it’s exactly the same process.

(b) Check your work by plotting the sum from \( n = -50 \) to 50 over the range \( t = -2 \) to 2. (Hint, it should look like the plot above.)
3. (8 points) Imagine that I have a really, really long coaxial cable, and I am sending a signal from the previous problem down that cable. At some point, the signal goes through a high-pass filter consisting of a capacitor and a resistor (as shown in the figure below). Then it passes through more cable, and eventually reaches an oscilloscope where the arriving signal is displayed.

(a) Since this is a linear system, let’s think of just one component in our sum at a time. If I just send a wave $V_0 e^{i\omega t}$ into the high pass filter, what comes out will still oscillate at the same frequency, but it will have a different amplitude and a phase shift. After the filter, the signal will look like $A_\omega V_0 e^{i\omega t}$ where $A_\omega$ is a constant for any given $\omega$. Note that since this is a complex wave, we can treat the capacitor as a resistor with an impedance of $1/j\omega$ and we won’t have to solve any differential equations. Use this to find $A_\omega$. (Side note - remember that any complex number can be written as $\alpha e^{i\phi}$ where $\alpha$ and $\phi$ are real. So multiplying by a complex constant can create both an amplitude change and a phase shift.)

(b) Now adjust each term from part (a) in the previous problem according to what we found in part (a) of this problem to find what the signal will be like after the high-pass filter.

(c) Plot your results from (b) - the wave after the filter. Only sum from $n = -50$ to $n = 50$, and assume that $R = C = 1$.

Here’s some background info for you on Discrete Fourier Transforms or DFTs. This info relates to the last half of this assignment.

When we evaluate real signals in a lab, the instruments we use (computers, oscilloscopes, etc.) generally only take data at discrete times. So rather than a continuous function, we have a list of numbers taken at specific times. We can still write our data as a sum of sines and cosines or complex exponentials, but because we have discrete points, there will be a natural limit to the highest frequency in that sum. 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
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 (known as the Nyquist frequency, \( \nu_{\text{Nyquist}} \)), 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 \( \nu_{\text{max}} = \nu_{\text{Nyquist}} = 1/2\tau \). So when I write my data as a sum of sines and cosines or complex exponentials, there will be a finite number of terms in my sum.

The next problem that we have with “real data” is that we only take data for a finite amount of time. As such, we never really know if the wave is truly periodic. So in order to perform a Fourier transform, we have to assume something about what the signal does outside of our sample window. The normal thing to do is to assume that it is periodic.

Because of these limitations, while the spectrum we get is useful, we have to remember that it isn’t an “exact” representation of the thing that we measured. Right at the time of a data point the series will give us an exact answer, but between data points the series will naturally interpolate, giving values that may or may not be close to the actual signal. But although the series can’t always give us accurate info between data points (how could it? - we didn’t give it any information about what happens between data points!), the Fourier series will be an exact representation of our data points.

The problem isn’t with the series, it’s with our data! We can exactly represent our incomplete data with a discrete Fourier transform, even if the frequencies in the transform do not accurately represent the thing which we measured when we took the data.

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

\[
f_k = \sum_{n=-N/2}^{N/2} c_n e^{2\pi i nk / N}.
\]

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

\[
f(t) = \sum_{n=-N/2}^{N/2} c_n e^{i\omega_0 t}.
\]

The expression that tells us the values of the \( c_n \) coefficients is\(^2\)

\[
c_n = \frac{1}{N} \sum_{k=0}^{N-1} f_k e^{-2\pi i nk / N}.
\]

The process of finding these coefficients and writing our data points as a sum is known as a “discrete Fourier transform,” or DFT. Now, on with the homework . . .

\(^2\)Most people don’t put the 1/N term in the transform, but instead put it into the inverse transform (and some, for the sake of symmetry, put a 1/\( \sqrt{N} \) in both the transform and the inverse transform). But if you do this, the \( c_n \) terms don’t actually tell you the amplitude of a given frequency. I’m not exactly sure why they do this, but I suspect it is because most software that performs Fourier transforms are set up to use integers rather than floating point numbers (for increased speed). If you wait until the end to divide by \( N \), that would reduce round-off error. Since we’re more interested in understanding than computer algorithm speed and accuracy, I’ll stick to the more intuitive form, which is more like the Fourier transforms we’ve seen before. Also note that most DFTs have \( n \) go from 0 to \( N - 1 \), rather than \( -N/2 \) to \( N/2 \). This works, and makes the transform and inverse transform look more alike. But once again, for the sake of understanding what is going on, I’ll stick with the latter.
4. (8 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. If you do this, the rest of the problems should go very quickly.)

(a) Define an array of numbers \( f_k = \sin(16\pi k/N) \) where \( N = 100 \) and \( k = 0, 1, 2, 3, \ldots, 99 \). Plot \( f_k \) and attach the plot to your homework.

(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. What are the values of \( n \) for which \( c_n \) should be non-zero?

(c) Have the computer calculate an 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) Now do the inverse transform, and plot the real and imaginary parts of the resulting \( f_k \) array.

5. (6 points) Now let \( f_k = \sin(14.7\pi k/N) \).

(a) Plot \( f_k \).

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

6. (4 points) Now let’s see what happens when we get frequencies close to or past half the sample 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 \( f_k = \sin(210\pi k/N) \). 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.