Physics 411: Homework 6

This homework is due in class on **Tuesday, March 12**, which is the first class after winter break. Example solutions to these problems will be handed out that same day (to allow you to use them in the second midterm, if you wish) and for that reason **no late homeworks** will be accepted. So if you cannot turn in this homework in class on March 12, please be sure to do so before class, not after.

Have a good break!

1. **Fourier transforms of simple functions:** Calculate the coefficients in the discrete Fourier transforms of the following periodic functions sampled at $N = 1000$ evenly spaced points, then make a plot of their magnitudes, similar to the plot shown in Fig. 7.4 in the book:

   (a) A single cycle of a square wave with amplitude 1

   (b) The sawtooth wave $y_n = n$

   (c) The modulated sine wave $y_n = \sin(\pi n/N)\sin(20\pi n/N)$

   ☑ **For full credit** turn in your program from part (a) and printouts of your three plots.

2. **Fourier transforms of real-life waveforms:** From the course web site download the files `piano.txt` and `trumpet.txt`, which each contain a waveform for a single note, played on, respectively, a piano and a trumpet. (Note: The files are not in the zip file that you downloaded at the start of the semester—you'll have to download them separately from the web site.) Write a program that loads a waveform, plots it, then calculates its discrete Fourier transform and plots the magnitudes of the first 10 000 coefficients. Apply your program to the piano and trumpet waveforms and discuss briefly what you observe about the piano and trumpet from the plots of Fourier coefficients.

   ☑ **For full credit** turn in your plots of the Fourier coefficients for each of the two waveforms, along with your discussion of what you observed.

3. **Detecting periodicity:** In the on-line resources there is a file called `sunspots.txt`, which contains the observed number of sunspots on the Sun for each month since January 1749. You previously made a visualization of the data in this file for Homework 2.

   (a) Write a short program (or dig up your old one from Homework 2) that reads the data in the file and makes a graph of sunspots as a function of time. You should see that the number of sunspots has fluctuated on a regular cycle for as long as observations have been recorded. Make an estimate of the length of the cycle in months.

   (b) Modify your program to calculate the Fourier transform of the sunspot data and then make a graph of the magnitude squared $|c_k|^2$ of the Fourier coefficients as a function of k—also called the *power spectrum* of the sunspot signal. You should see that there is a noticeable peak in the power spectrum at a nonzero value of k. The appearance of this peak tells us that there is one frequency in the Fourier series that

has a higher amplitude than the others around it—meaning that there is a large sine-wave term with this frequency, which corresponds to the periodic wave you can see in the original data.

(c) Find the approximate value of k to which the peak corresponds. What is the period of the sine wave with this value of k? You should find that the period corresponds roughly to the length of the cycle that you estimated in part (a) above.

This kind of Fourier analysis is a sensitive method for detecting periodicity in signals. Even in cases where it is not clear to the eye that there is a periodic component to a signal, it may still be possible to find one using a Fourier transform.

☑ **For full credit** turn your program and plot from part (b) and your results from part (c).

4. **Fourier filtering and smoothing:** In the on-line resources you'll find a file called `dow.txt`. It contains the daily closing value for each business day from late 2006 until the end of 2010 of the Dow Jones Industrial Average, which is a measure of average prices on the US stock market.

   Write a program to do the following:

   (a) Read in the data from `dow.txt` and plot them on a graph.

   (b) Calculate the coefficients of the discrete Fourier transform of the data using the function `rfft` from `numpy.fft`, which produces an array of $\frac{1}{2}N + 1$ complex numbers.

   (c) Now set all but the first 10% of the elements of this array to zero (i.e., set the last 90% to zero but keep the values of the first 10%).

   (d) Calculate the inverse Fourier transform of the resulting array, zeros and all, using the function `irfft`, and plot it on the same graph as the original data. You may need to vary the colors of the two curves to make sure they both show up on the graph. Comment on what you see. What is happening when you set the Fourier coefficients to zero?

   (e) Modify your program so that it sets all but the first 2% of the coefficients to zero and run it again.

   ☑ **For full credit** turn a printout of your program, a plot showing all three curves on the same axes, and your answer to the question in part (d).

5. **Extra credit:** This problem is optional, for those who want an extra challenge (and extra credit). It is pretty difficult, but you can learn a lot by completing it.

   (a) Write your own program to compute the fast Fourier transform of a given set of N samples, in the case where N is a power of two, based on the formulas given in Section 7.4.1. As a test of your program, you can use it to calculate the Fourier transform of the data in the file `pitch.txt`, which can be found in the on-line resources, and which is shown in Fig. 7.3 in the book (and its transform is shown in Fig. 7.4).

You will have to calculate the coefficients $E_k^{(m,j)}$ from Eq. (7.43) for all levels m, which means that first you will have to plan how the coefficients will be stored. Since, as we have seen, there are exactly N of them at every level, one way to do it would be to create a two-dimensional complex array of size $N \times (1 + \log_2 N)$, so that it has N complex numbers for each level from zero to $\log_2 N$. Then within level m you have 2^m individual transforms denoted by $j = 0 \ldots 2^m - 1$, each with $N/2^m$ coefficients indexed by k. A simple way to arrange the coefficients would be to put all the $k = 0$ coefficients in a block one after another, then all the $k = 1$ coefficients, and so forth. Then $E_k^{(m,j)}$ would be stored in the $j + 2^m k$ element of the array.

This method has the advantage of being quite simple to program, but the disadvantage of using up a lot of memory space. The array contains more than $N \log_2 N$ complex numbers, and a complex number takes up 16 bytes of memory. So if you had to do a large Fourier transform of, say, $N = 10^8$ numbers, it would take $16N \log_2 N \simeq 42$ gigabytes of memory, which is more than most computers have.

An alternative approach is to notice that we do not really need to store all of the coefficients. At any one point in the calculation we only need the coefficients at the current level and the previous level (from which the current level is calculated). If one is clever one can write a program that uses only two arrays, one for the current level and one for the previous level, each consisting of N complex numbers. Then our transform of 10^8 numbers would require less than four gigabytes, which is fine on most computers.

(b) Use your program to duplicate the plot shown in Fig. 7.4. Confirm that your plot looks the same as the one in the figure.

☑ **To receive a substantial amount of extra credit** turn a printout of your program and your final plot.