Physics 411: Homework 6

1. **Fourier transforms of example waveforms:** In the on-line resources you will find files called `piano.txt` and `trumpet.txt`, which contain data representing the waveform of a single note, played on, respectively, a piano and a trumpet.

   (a) Write a program that loads a waveform from one of these files, plots it, then calculates its discrete Fourier transform and plots the magnitudes of the first 10 000 coefficients in a manner similar to Fig. 7.4 in the book. Note that you will have to use a fast Fourier transform for the calculation because there are too many samples in the files to do the transforms the slow way in any reasonable amount of time.

   Apply your program to the piano and trumpet waveforms and discuss briefly what one can conclude about the sound of the piano and trumpet from the plots of Fourier coefficients.

   (b) Both waveforms were recorded at the industry-standard rate of 44 100 samples per second and both instruments were playing the same musical note when the recordings were made. From your Fourier transform results calculate what note they were playing. (Hint: The musical note middle C has a frequency of 261 Hz.)

   ✓ **For full credit** turn in your plots of the Fourier coefficients for each of the two waveforms, your conclusions about what you observed, and your calculation and results from part (b).

2. **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 the calculation again. Make a plot that shows all three curves on the same axes—the raw data, the 10% curve, and the 2% curve.

The particular data set studied here is special in one sense: the value of the Dow at the end of the period was almost the same as at the start, so the function is, roughly speaking, periodic. In the on-line resources there is another file called `dow2.txt`, which also contains data on the Dow but for a different time period, from 2004 until 2008. Over this period the value changed considerably from a starting level around 9000 to a final level around 14000.

(f) Modify your program to read the data from `dow2.txt` and perform the Fourier analysis again, plotting just two curves this time, for the raw data and the "2%" version. You should see that there is now an additional artifact in the 2% curve. At the beginning and end of the plot you should see large deviations away from the true function. These occur because the function is required to be periodic—its last value must be the same as its first—so it needs to deviate substantially from the correct value to make the two ends of the function meet. In some situations (including this one) this behavior is unsatisfactory. We would prefer not to introduce artifacts of this kind.

(g) Modify your program to repeat the same analysis using a discrete cosine transform. Transform the data and again discard all but the first 2% of the coefficients, then invert the transform and plot the result. You should see a significant improvement, with less distortion of the function at the ends of the interval. This occurs because, as we discussed in class, the cosine transform does not force the value of the function to be the same at both ends. Hint: You can use the functions from the file `dcst.py` in the on-line resources to perform the transforms if you wish—just place the file in your working folder and then import the functions using a `from` statement. The function `dct` does the cosine transform and `idct` does the inverse transform.

It is because of the artifacts introduced by the strict periodicity of the DFT that the cosine transform is favored for many technological applications, such as audio compression. The artifacts can degrade the sound quality of compressed audio and the cosine transform generally gives better results.

☑ **For full credit** turn in a printout of your program from part (e), a plot showing all three curves it calculates on the same axes (raw data, 10%, and 2%), and your answer to the question in part (d). Also turn in your two plots from parts (f) and (g) showing your analysis of the second data set using the DFT and the DCT respectively (or you can combine (f) and (g) into a single plot if you prefer).

3. **Image compression:** In this problem you will write your own program to do JPEG-style compression of a photographic image.

   First download the file `house.txt` from the course web site. (This file is not in the on-line resources—you'll find it separately listed on the web site.) The file contains data for a picture of a house in simple grid form—a two-dimensional array of numbers representing the intensity of pixels in the image.

   (a) Write a Python program that reads the data in the file into a two-dimensional array and then makes a density plot of the array, showing the picture on the screen. You should use the gray-scale color scheme for your density plot, so you get a sensible looking black-and-white photograph.

(b) Now create another two-dimensional array of floats of the same size as the picture array and initially empty. Go through the picture array in 16×16 blocks and perform a 2D discrete cosine transform of the data in each block, producing a 16×16 array of (real) Fourier coefficients, and then store those coefficients in the corresponding block of the new array. You can perform the DCTs using the function `dct2` from the file `dcst.py`. To get the 16×16 blocks you'll need to do two-dimensional "slicing" on the arrays—see page 67 in the book if you want a reminder of how to do this. When you're finished with all the blocks, you will have a new array of the same size as the old one, entirely full of Fourier coefficients.

(c) Now go through the Fourier coefficients one by one and set to zero every coefficient whose *absolute value* is less than 10. In other words every coefficient in the range from -10 to $+10$ should get set to zero.

(d) When we send a picture over the Internet, we transmit the Fourier coefficients, not the picture itself, and we only need to transmit the coefficients that are nonzero. Count how many coefficients get set to zero in your calculation and use this to calculate and print a figure for how much you have compressed the image—how much smaller is the set of numbers you would have to send over the Internet than the original set from the file `house.txt`? This figure is called the *compression ratio*.

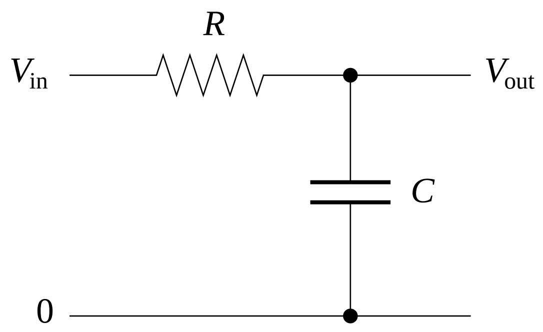
(e) When the Fourier coefficients are received at the other end, the receiver performs an inverse transform to recover the image. Although we are not actually transmitting our picture in this case, we can still perform this second part of the calculation to see what we would get. Add lines to your program to go once more through the array of Fourier coefficients in 16×16 blocks and perform an inverse 2D DCT on each one, storing the results back in the original data array again (or in a third, new array, if you prefer). This is the "decompression" of the image. You can use the function `idct2` for the inverse DCTs.

(f) Make a density plot of the decompressed image. You should find that it is essentially indistinguishable from the original picture, even though the image was compressed quite lot—a significant number of the Fourier coefficients were discarded by setting them to zero.

(g) Increase the threshold value below which the coefficients get set to zero. Instead of 10, try 20, or 50, or 100 or more. See how large a compression ratio you can achieve and still have the picture look pretty much the same.

☑ **For full credit** turn in a printout of your final program and the "before" and "after" images it produces, along with your answer for part (d) and your findings from part (g).

4. **A low-pass filter:** Here is a simple electronic circuit with one resistor and one capacitor:



This circuit acts as a low-pass filter: you send a signal in on the left and it comes out filtered on the right.

Using Ohm's law and the capacitor law and assuming that the output load has very high impedance, so that a negligible amount of current flows through it, we can write down the equations governing this circuit as follows. Let I be the current that flows through R and into the capacitor, and let Q be the charge on the capacitor. Then:

$$IR = V_{\text{in}} - V_{\text{out}}, \qquad Q = CV_{\text{out}}, \qquad I = \frac{\mathrm{d}Q}{\mathrm{d}t}.$$

Substituting the second equation into the third, then substituting the result into the first equation, we find that $V_{\text{in}} - V_{\text{out}} = RC\,(\mathrm{d}V_{\text{out}}/\mathrm{d}t)$, or equivalently

$$\frac{\mathrm{d}V_{\text{out}}}{\mathrm{d}t} = \frac{1}{RC}\big(V_{\text{in}} - V_{\text{out}}\big).$$

(a) Write a program to solve this equation for $V_{\text{out}}(t)$ using the fourth-order Runge–Kutta method when the input signal is a square-wave with frequency 1 and amplitude 1:

$$V_{\text{in}}(t) = \begin{cases} 1 & \text{if } \lfloor 2t \rfloor \text{ is even,} \\ -1 & \text{if } \lfloor 2t \rfloor \text{ is odd,} \end{cases} \tag{1}$$

where $\lfloor x \rfloor$ means x rounded down to the next lowest integer. Use the program to make plots of the output of the filter circuit from $t = 0$ to $t = 10$ when $RC = 0.01$, 0.1, and 1, with initial condition $V_{\text{out}}(0) = 0$. You will have to make a decision about what value of h to use in your calculation. Small values give more accurate results, but the program will take longer to run. Try a variety of different values and choose one for your final calculations that seems sensible to you.

(b) Based on the graphs produced by your program, describe what you see and explain what the circuit is doing.

A program similar to the one you wrote is running inside most sound systems and music players, to create the effect of the "bass" control. In the old days, the bass control on a stereo would have been connected to a real electronic low-pass filter in the amplifier circuitry, but these days there is just a computer processor that simulates the behavior of the filter in a manner similar to your program.

☑ **For full credit** turn in a printout of your program and the plots it produces, along with your answer to part (b).

5. **The Lotka–Volterra equations:** The Lotka–Volterra equations are a mathematical model of predator–prey interactions between biological species. Let two variables x and y be proportional to the size of the populations of two species, traditionally called "rabbits" (the prey) and "foxes" (the predators). You could think of x and y as being the population in thousands, say, so that $x = 2$ means there are 2000 rabbits. Strictly the only allowed values of x and y would then be multiples of 0.001, since you can only have whole numbers of rabbits or foxes. But 0.001 is a pretty close spacing of values, so it's a decent approximation to treat x and y as continuous real numbers so long as neither gets very close to zero.

   In the Lotka–Volterra model the rabbits reproduce at a rate proportional to their population, but are eaten by the foxes at a rate proportional to both their own population and the population of foxes:
$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy,$$
   where α and β are constants. At the same time the foxes reproduce at a rate proportional the rate at which they eat rabbits—because they need food to grow and reproduce—but also die of old age at a rate proportional to their own population:
$$\frac{\mathrm{d}y}{\mathrm{d}t} = \gamma xy - \delta y,$$
   where γ and δ are also constants.

   (a) Write a program to solve these equations using the fourth-order Runge–Kutta method for the case $\alpha = 1$, $\beta = \gamma = 0.5$, and $\delta = 2$, starting from the initial condition $x = y = 2$. Have the program make a graph showing both x and y as a function of time on the same axes from $t = 0$ to $t = 30$. (Hint: Notice that the differential equations in this case do not depend explicitly on time t—in vector notation, the right-hand side of each equation is a function $f(\mathbf{r})$ with no t dependence. You may nonetheless find it convenient to define a Python function `f(r,t)` including the time variable, so that your program takes the same form as programs given in the book.)

   (b) Describe in words what is going on in the system, in terms of rabbits and foxes.

☑ **For full credit** turn in a printout of your program and the graph it produces, along with your answer to part (b).