Physics 411: Homework 3

1. **Quadratic equations:**

   (a) Write a program that takes as input three numbers, a, b, and c, and prints out the two solutions to the quadratic equation $ax^2 + bx + c = 0$ using the standard formula

   $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

   Use your program to compute the solutions of $0.001x^2 + 1000x + 0.001 = 0$.

   (b) There is another way to write the solutions to a quadratic equation. Multiplying top and bottom of the solution above by $-b \mp \sqrt{b^2 - 4ac}$, show that the solutions can also be written as

   $$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}.$$

   Add further lines to your program to print out these values in addition to the earlier ones and again use the program to solve $0.001x^2 + 1000x + 0.001 = 0$. What do you see? How do you explain it?

   (c) Using what you have learned, modify your program so that it calculates both roots of a quadratic equation accurately in all cases.

☑ **For full credit** turn in your answers to part (b), a copy of your final program, and a printout of it in action, showing the solution of the equation $0.001x^2 + 1000x + 0.001 = 0$.

This is a good example of how computers don't always work the way you expect them to. If you simply apply the standard formula for the quadratic equation, the computer will sometimes get the answer wrong. In practice the method you have worked out here is the correct way to solve a quadratic equation on a computer, even though it's more complicated than the standard formula. If you were writing a program that involved solving many quadratic equations this method might be a good candidate for a user-defined function.

2. **Calculating derivatives:** Suppose we have a function $f(x)$ and we want to calculate its derivative at a point x. We can do that with pen and paper if we know the mathematical form of the function, or we can do it on the computer by making use of the definition of the derivative:

   $$\frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}.$$

   On the computer we can't actually take the limit as δ goes to zero, but we can get a reasonable approximation just by making δ small.

   (a) Write a program that defines a function `f(x)` returning the value $x(x-1)$, then calculates and prints the derivative of the function at the point $x = 1$ using the formula above with $\delta = 10^{-2}$. Calculate the true value of the same derivative analytically and compare with the answer your program gives. The two will not agree perfectly. Why not?

(b) Repeat the calculation for $\delta = 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}$, and 10^{-14}. You should see that the accuracy of the calculation initially gets better as δ gets smaller, but then gets worse again. Why is this?

For full credit, turn in a printout of your program, the results from the various calculations, and your answer to the question in part (b).

We will look at numerical derivatives in more detail later in the course, when we will study techniques for dealing with these issues.

3. **Simpson's rule:**

   (a) Write a program to calculate an approximate value for the integral $\int_0^2 (x^4 - 2x + 1)\,\mathrm{d}x$ from Example 5.1, but using Simpson's rule with 10 slices instead of the trapezoidal rule.

   (b) Run the program and compare your result to the known correct value of 4.4. What is the fractional error on your calculation?

   (c) Modify the program to use a hundred slices instead, then a thousand. Note the improvement in the result. How do the results compare with those from Example 5.1 for the trapezoidal rule with the same number of slices?

   For full credit turn in a printout of your program, plus your results and a brief discussion of how they compare with the trapezoidal rule.

4. **Adaptive integration:** Write a program that uses the adaptive trapezoidal rule method of Section 5.3 and Eq. (5.30) to calculate the value of the integral

$$I = \int_0^1 \sin^2 \sqrt{100x}\,\mathrm{d}x$$

   to an approximate accuracy of $\epsilon = 10^{-6}$ (i.e., correct to six digits after the decimal point). Start with one single integration slice and work up from there to two, four, eight, and so forth. Have your program print out the number of slices, its estimate of the integral, and its estimate of the error on the integral, for each value of the number of slices N, until the target accuracy is reached. (Hint: You should find the result is around $I = 0.45$.)

   Extra credit: *This part of the problem is optional, for extra credit.* Modify your program to evaluate the same integral using the Romberg integration technique described in Section 5.4, for the same series of values of N. Have your program print out a triangular table of values, as on page 157, of all the Romberg estimates of the integral.

   For full credit turn in a printout of your program and a printout of it in action, clearly showing the output values. If you did the extra credit part, turn in that program too, plus a printout of it in action.

5. **The diffraction limit of a telescope:** Our ability to resolve detail in astronomical observations is limited by the diffraction of light in our telescopes. Light from stars can be

treated effectively as coming from a point source at infinity. When such light, with wavelength λ, passes through the circular aperture of a telescope (which we'll assume to have unit radius) and is focused by the telescope in the focal plane, it produces not a single dot, but a circular diffraction pattern consisting of central spot surrounded by a series of concentric rings. The intensity of the light in this diffraction pattern is given by

$$I(r) = \left(\frac{J_1(kr)}{kr}\right)^2,$$

where r is the distance in the focal plane from the center of the diffraction pattern, $k = 2\pi/\lambda$, and $J_1(x)$ is a Bessel function. The Bessel functions $J_m(x)$ are given by

$$J_m(x) = \frac{1}{\pi}\int_0^\pi \cos(m\theta - x\sin\theta)\,\mathrm{d}\theta,$$

where m is a nonnegative integer and $x \geq 0$.

(a) Write a Python function `J(m,x)` that calculates the value of $J_m(x)$ using Simpson's rule with $N = 1000$ points. Use your function in a program to make a plot, on a single graph, of the Bessel functions J_0, J_1, and J_2 as a function of x from $x = 0$ to $x = 20$.

(b) Make a second program that makes a density plot of the intensity of the circular diffraction pattern of a point light source with $\lambda = 500\,\text{nm}$, in a square region of the focal plane, using the formula given above. Your picture should cover values of r from zero up to about $1\,\mu\text{m}$.

Hint 1: You may find it useful to know that $\lim_{x\to 0} J_1(x)/x = \frac{1}{2}$.

Hint 2: If you have a variable in your program that represents the wavelength λ, do not call it "`lambda`". If you do you will get an error message and the program will not run, because the word "lambda" has a special meaning in the Python language and cannot be used as a variable name, just as words like "for" and "if" cannot be used as variable names. (See footnote 5 on page 13 of the book.) The names of other Greek letters—alpha, beta, gamma, and so on—are allowed as variable names.

Hint 3: The central spot in the diffraction pattern is so bright that it may be difficult to see the rings around it on the computer screen. If you run into this problem a simple way to deal with it is to use one of the other color schemes for density plots described in Section 3.3 of the book. The "`hot`" scheme works well. For a more sophisticated solution to the problem, the `imshow` function has an additional argument `vmax` that allows you to set the value that corresponds to the brightest point in the plot. For instance, if you say "`imshow(x,vmax=0.1)`", then elements in `x` with value 0.1, or any greater value, will produce the brightest (most positive) color on the screen. By lowering the `vmax` value, you can reduce the total range of values between the minimum and maximum brightness, and hence increase the sensitivity of the plot, making subtle details visible. For this exercise I found that a value `vmax=0.01` worked well for me.

☑ **For full credit** turn in a printout of your final program from part (b) and printouts of your two figures.