Physics 411: Homework 2

1. **The semiempirical mass formula again:** Last week you solved a problem using the semiempirical mass formula for calculating the nuclear binding energy of an atomic nucleus. In this problem you will use some of the additional features of Python that we discussed this week to extend that calculation. You may find it useful to dig up your program from last week and use that as a starting point for this problem.

   Recall that the semi-empirical mass formula gives an approximate value for the binding energy B of a nucleus with atomic number Z and mass number A:

$$B = a_1 A - a_2 A^{2/3} - a_3 \frac{Z^2}{A^{1/3}} - a_4 \frac{(A-2Z)^2}{A} + \frac{a_5}{A^{1/2}},$$

   where, in units of millions of electron volts, the constants are $a_1 = 15.67$, $a_2 = 17.23$, $a_3 = 0.75$, $a_4 = 93.2$, and

$$a_5 = \begin{cases} 12.0 & \text{if } Z \text{ and } A-Z \text{ are both even,} \\ -12.0 & \text{if } Z \text{ and } A-Z \text{ are both odd,} \\ 0 & \text{otherwise.} \end{cases}$$

   (a) Write a program, or modify your previous one, to calculate and print out the binding energy per nucleon B/A for values of Z and A entered by the user. Test your program for $A = 58$ and $Z = 28$. You should get an answer around 8.5 MeV per nucleon.

   (b) Now modify your program so that it takes as input just a single value of the atomic number Z and then goes through all values of A from $A = Z$ to $A = 3Z$, to find the one that has the largest binding energy per nucleon. This is the most stable nucleus with the given atomic number. Have your program print out the value of A for this most stable nucleus and the value of the binding energy per nucleon.

   (c) Modify your program again so that, instead of taking Z as input, it runs through all values of Z from 1 to 100 and prints out the most stable value of A for each one. At what value of Z does the maximum binding energy per nucleon occur? This value represents the highest Z that can be generated by nuclear fusion in stars—fusion of nuclei to form heavier atoms will absorb energy, not generate it. Hint: the true answer, in real life, is $Z = 26$, which is iron—there are no naturally occurring elements higher up the periodic table than iron, except in trace amounts. You should find that the semiempirical mass formula gets the answer roughly right, but not exactly.

   ☑ **For full credit** turn in a printout of the final version of your program, plus a printout of it in action, showing the results it gives, along with your answer to the question in the last part.

2. **The Madelung constant:** In condensed matter physics the Madelung constant gives the total electric potential felt by an atom in a solid. It depends on the charges on the other atoms nearby and their locations. Consider for instance solid sodium chloride—table salt. The sodium chloride crystal has atoms arranged in a cubic lattice, but with alternating sodium and chlorine atoms, the sodium ones having a single positive charge $+e$ and the chlorine ones a single negative charge $-e$, where e is the charge on the electron. If we label each position on the lattice by three integer coordinates (i, j, k), then the sodium atoms fall at positions where $i + j + k$ is even, and the chlorine atoms at positions where $i + j + k$ is odd.

   Consider a sodium atom at the origin, $i = j = k = 0$, and let us calculate the Madelung constant. If the spacing of atoms on the lattice is a, then the distance from the origin to the atom at position (i, j, k) is

   $$\sqrt{(ia)^2 + (ja)^2 + (ka)^2} = a\sqrt{i^2 + j^2 + k^2},$$

   and the potential created at the origin by such an atom is

   $$V(i, j, k) = \pm \frac{e}{4\pi\epsilon_0 a\sqrt{i^2 + j^2 + k^2}},$$

   with ϵ_0 being the permittivity of the vacuum and the sign of the expression depending on whether $i + j + k$ is even or odd. The total potential felt by the sodium atom is then the sum of this quantity over all atoms. Let us assume a cubic box around the sodium at the origin, with L atoms in all directions. Then

   $$V_{\text{total}} = \sum_{\substack{i,j,k=-L \\ (\text{not } i=j=k=0)}}^{L} V(i, j, k) = \frac{e}{4\pi\epsilon_0 a} M,$$

   where M is the Madelung constant, at least approximately—technically the Madelung constant is the value of M when $L \to \infty$, but one can get a good approximation just by using a large value of L.

   Write a program to calculate and print the Madelung constant for a sodium atom in sodium chloride. Use as large a value of L as you can, while still having your program run in reasonable time—say in a minute or less.

   ☑ **For full credit** turn in a copy of your program and a printout of it in action, showing the value it calculates.

3. **Prime numbers:** We saw in class a program for finding prime numbers but that program was not particularly efficient: it checked each number to see if it was divisible by any number less than it. We can develop a much faster program for prime numbers by making use of the following observations:

   (a) A number n is prime if it has no prime factors less than n. Hence we only need to check if it is divisible by other primes.

(b) If a number n is non-prime, having a factor r, then $n = rs$, where s is also a factor. If $r \geq \sqrt{n}$ then $n = rs \geq \sqrt{n}s$, which implies that $s \leq \sqrt{n}$. In other words, any non-prime must have factors (and hence also prime factors) less than or equal to $\sqrt{n}$.

Thus to determine if a number is prime we have to check its prime factors only up to and including $\sqrt{n}$—if there are none then the number is prime.

Write a Python program that prints all the primes up to 10 000. Create a list to store the primes, which starts out with just the one prime number 2 in it, then for each number n from 3 to 10 000 check whether the number is divisible by any of the primes in the list up to and including $\sqrt{n}$. As soon as you find a single prime factor you can stop checking the rest of them—you know n is not a prime. If you find no prime factors $\sqrt{n}$ or less then n is prime and you should add it to the list. You can print out the list all in one go at the end of the program, or you can print out the individual numbers as you find them.

✓ **For full credit** turn in a copy of your program along with the last five (largest) primes it finds. No need to turn in a printout of the whole list of primes—it would be very long.

4. **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 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.

5. **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 pencil 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:

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

On the computer we can't actually take the limit as ϵ goes to zero, but we can get a reasonable approximation just by making ϵ 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 $\epsilon = 10^{-2}$. 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 $\epsilon = 10^{-4}$, 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}, and 10^{-14}. You should see that the accuracy of the calculation initially gets better as ϵ 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 the last part.

We will look at numerical derivatives in more detail later in the course, when we will study techniques for dealing with these issues and maximizing the accuracy of our calculations.