Exam #1

Computational Physics Lab

Instructor: Prof. Sean Dobbs   ¶

February 24, 2020

*Due by 11 PM, March 4th**

All problems must be solved without consulting others or copying from any source, except that you may use programs from the textbook or programs from the exercises as a starting point.

Submission

For each problem, you will write one or more python programs. These programs should follow the Python coding and formatting conventions outlined for the course. Many problems will have additional questions to answer, or will ask for a record of the terminal showing the output when the program is run with various arguements. These answers can be given either in block comments in the prologue of the corresponding assignment, or in clearly labeled text files. The terminal output should contain some information on the username and machine you are running on (this will often be in the command prompt).

You will be evaluated based on the files contained in your remote GitHub repository at the due date. You should and commit all the files you created to your local repository on a regular basis. You should do this by adding any new or modified files using "git add", and then finalizing changes using "git commit". Remember that "git status" will give you information on which files in your repository. have been changed. Remember to add short, but useful comments when performing a commit. When you are finished with the exercise, push the current status of your local repository to the remote repository on GitHub using the command "git push -u". You are encouraged to push your local files to the remote repository periodically before you are done, and certainly well before the deadline, if possible.

1. Special relativity

A spaceship travels from Earth in a straight line at a (relativistic) speed v to another planet x light years away. Write a program to ask the user for the value of x and the speed v as a fraction of the speed of light, then print out the time in years that the spaceship takes to reach its destination (a) in the rest frame of an observer on Earth and (b) as perceived by a passenger on board the ship. Use your program to calculate the answers for a planet 10 light years away with $v = 0.99c$.

For full credit turn in your program, and a print out of the results.

2. Prime numbers

The program in Example 2.8 is not a very efficient way of calculating prime numbers: it checks each number to see if it is divisible by any number less than it. We can develop a much faster program for prime numbers by making use of the following observations:

1) 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.

2) 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.

3) If we find even a single prime factor less than $\sqrt{n}$ then we know that the number is non-prime, and hence there is no need to check any further---we can abandon this number and move on to something else.

Write a Python program that finds all the primes up to ten thousand. 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 your program, and a print out of the results.

3. Quantum uncertainty in the harmonic oscillator

In units where all the constants are 1, the wavefunction of the nth energy level of the one-dimensional quantum harmonic oscillator - i.e., a spinless point particle in a quadratic potential well - is given by

$$\psi_n(x) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} e^{-x^2/2} H_n(x),$$

for $n = 0 \ldots \infty$, where $H_n(x)$ is the nth Hermite polynomial. Hermite polynomials satisfy a relation somewhat similar to that for the Fibonacci numbers, although more complex:

$$H_{n+1}(x) = 2x H_n(x) - 2n H_{n-1}(x).$$

The first two Hermite polynomials are $H_0(x) = 1$ and $H_1(x) = 2x$.

a) Write a user-defined function `H(n,x)` that calculates $H_n(x)$ for given x and any integer $n \geq 0$. Use your function to make a plot that shows the harmonic oscillator wavefunctions for $n = 0, 1, 2,$ and 3, all on the same graph, in the range $x = -4$ to $x = 4$. Hint: There is a function `factorial` in the `math` package that calculates the factorial of an integer. You can also use the function that you wrote in Exercise 2.X.

b) Make a separate plot of the wavefunction for $n = 30$ from $x = -10$ to $x = 10$. Hint: If your program takes too long to run in this case, then you're doing the calculation wrong - the program should take only a second or so to run.

c) The quantum uncertainty of a particle in the nth level of a quantum harmonic oscillator can be quantified by its root-mean-square position $\sqrt{\langle x^2 \rangle}$, where

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} x^2 |\psi_n(x)|^2 \, dx.$$

Write a program that evaluates this integral using Gaussian quadrature on 100 points and then calculates the uncertainty (i.e., the root-mean-square position of the particle) for a given value of n. Use your program to calculate the uncertainty for $n = 5$. You should get an answer in the vicinity of $\sqrt{\langle x^2 \rangle} = 2.3$.

For full credit turn in your program, your graphs for parts (a) and (b), and your result for part (c).

4. The Stefan-Boltzmann constant

The Planck theory of thermal radiation tells us that in the (angular) frequency interval ω to $\omega + d\omega$, a black body of unit area radiates electromagnetically an amount of thermal energy per second equal to $I(\omega) \, d\omega$, where

$$I(\omega) = \frac{\hbar}{4\pi^2 c^2} \frac{\omega^3}{(e^{\hbar\omega/k_B T} - 1)}.$$

Here $\hbar$ is Planck's constant over 2π, c is the speed of light, and k_B is Boltzmann's constant.

a) Show that the total energy per unit area radiated by a black body is

$$W = \frac{k_B^4 T^4}{4\pi^2 c^2 \hbar^3} \int_0^\infty \frac{x^3}{e^x - 1} \, dx.$$

b) Write a program to evaluate the integral in this expression. Explain what method you used, and how accurate you think your answer is.

c) Even before Planck gave his theory of thermal radiation around the turn of the 20th century, it was known that the total energy W given off by a black body per unit area per second followed Stefan's law: $W = \sigma T^4$, where σ is the Stefan-Boltzmann constant. Use your value for the integral above to compute a value for the Stefan-Boltzmann constant (in SI units) to three significant figures. Check your result against the known value, which you can find in books or online. You should get good agreement.

For full credit turn in a your problem and your answers to each part.