Spring, 2019

Physics 480/905: Assignment #1

This HW assignment is designed to give you practice in the basic tasks from class, which we'll need repeatedly and build upon, and to verify to me that you can do them. Remember, I'll also upload some "hints and suggestions" on the course website for some of the problems.

The HW is due by the end of class on Friday Feb. 8. If you think you might have trouble finishing by the deadline, please let me know. I'm willing to have a little bit of flex if it ensures that you solve, or at least make a good-faith effort to solve, all the problems. You'll turn it in by sharing your GitHub username with me– see Session 4 Activities for instructions on creating a PHY480 repository for those who are new to GitHub.

Use C++ for the codes and gnuplot for the plots. *It is required that your code have appropriate comments.* Comment your codes with your name, email, AND revision history, as in the example codes from class.

The overall goal of these tasks is to teach/remind you of some basic programming and reinforce some of the issues stemming from the approximate representation of real numbers.

1. **Update to `area.cpp`.**

   Implement items 1 to 6 from the "to do" list in the `area.cpp` code from Activities 1 in a new code called `area_new.cpp`. If C++ is new to you, don't panic! We'll iterate until it makes sense. If you are an experienced C++ program, do item 7 (implement a Circle class) instead.

2. **Summing up vs. summing down.**

   This problem is taken from problem 3 in section 3.4 of the Landau–Paez *Computational Physics* text, which examines the summation of $1/n$. The analysis should be similar to the one on finding the roots of quadratic equations from class (you might find the quadratic_equation_2.cpp listing useful). Consider the two series for integer N:

   $$S^{(\text{up})} = \sum_{n=1}^{N} \frac{1}{n} \qquad S^{(\text{down})} = \sum_{n=N}^{1} \frac{1}{n}$$

   Mathematically they are finite and equivalent, because it doesn't matter in what order you do the sum. However, when you sum numerically, $S^{(\text{up})} \neq S^{(\text{down})}$ because of round-off error.

(a) Write a program (and a makefile) to calculate $S^{(\mathrm{up})}$ and $S^{(\mathrm{down})}$ in single precision as functions of N. Make sure you include appropriate comments and indent it consistently.

(b) Make a log–log plot of the difference divided by the sum (or average), i.e.,

$$\frac{|S^{(\mathrm{up})} - S^{(\mathrm{down})}|}{\frac{1}{2}(|S^{(\mathrm{up})}| + |S^{(\mathrm{down})}|)} ,$$

versus N and turn in a postscript or pdf file of the plot.

(c) Interpret the different regions of your graph (e.g., Are the calculations equally precise in some regions or is one way of doing the calculation always better? Is there a region where the error looks like power law and if so, what power? [You don't need to *explain* the power, just extract the value.] What happens for large values of N?) Explain qualitatively in your own words why the downward sum is more precise. Put these discussions in the comments of your code at the top.

3. **Spherical Bessel Functions.**

The goal here is to complete (some of) the Bessel function activities from Session 2. In particular,

(a) Turn in a code that generates the output file needed for Bessel 2, part 3.

(b) Turn in a postscript plot of the error vs. x made from that output file *with your interpretation of the different regions of the graph.* (Put your analysis in the comments at the top of your code from (a).)

(c) Add the GSL routine (from Bessel 3) to your code (so only one code is needed in the end), with a new column in the output file being $j_{10}(x)$ from GSL.

4. (BONUS) **Randomness of round-off errors.**

[This is not a required program, but I recommend giving it a try if you found the other parts easy.] In the book "Computational Physics", Landau and Paez say that the round-off errors in single-precision are distributed approximately randomly. Your task is to test whether this is true and what the distribution actually looks like. More specifically, if we generate a large set of $z_c = z(1+\epsilon)$ numbers from some sufficiently complex calculation (e.g., by taking the square root of a bunch of numbers), how are the different ϵ's distributed?

(a) Devise a scheme to test for the distribution of round-off errors.

(b) Carry out your scheme with a C++ program.

(c) Make appropriate plots to explore your results.