

## Physics 780.20: Assignment #1

This 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. Please ask questions! Because of the flooding emergency and the holiday, the revised schedule has Problem 1 due by midnight on Monday, January 16 (and we'll iterate until done) and the rest by midnight on Friday, January 20.

You are going to use Mercurial to “hand in” the assignment. Instructions on how to do this will be posted on the webpage. [Note: We're doing this for only the second time, so there will probably be glitches. But we'll work them out.]

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. Check the 780.20 webpage for suggestions and hints.

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. If you are an experienced programmer, I recommend that you try the bonus problem (which you can substitute for problem 3).

### 1. 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` printout useful). Consider the two series for integer  $N$ :

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

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 (with makefile) to calculate  $S^{(\text{up})}$  and  $S^{(\text{down})}$  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 versus  $N$  and turn in a postscript file of the plot.
- (c) Discuss the interpretation of the linear region on your graph and explain briefly why the downward sum is more precise. Put your discussion in the comments of the code.

## 2. 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 at the top of your code in the comments.)
- (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.

## 3. Update to `area.cpp`.

The idea here is to pick two (or more!) of the items from the “to do” list in the `area.cpp` code from Session 1 that you *don't* already know how to do (or haven't done before or not for a long time) and implement them in `area_new.cpp`.

## 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  $\epsilon$ '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.