PHY480/905: Activities 2

*Online handouts:* gnuplot mini-manual; gnuplot example; listings of several codes.

*Your goals (in order of priority):*

* If you have not turned in your Activity 1 worksheet, please do so ASAP via GitHub (ask if you need help setting up GitHub and adding me as a collaborator!)
* Look at a round-off error demonstration with quadratic equations
* Discuss a special-case problem to help understand summing in different orders
* Debug sample codes; make some plots with gnuplot
* Assess different methods of calculating spherical Bessel functions

Please work in pairs and briefly answer questions asked here on this sheet. The instructors will bounce around the room and answer questions (don't be shy to ask about anything!).

You should go to your working directory (see Activities 1 for creating one), download session02.zip from the course homepage and unzip it in your directory.

Round-Off Errors with Quadratic Equations

This section assumes you have read the background notes for Session 2, which describe round-off errors and the quadratic equation.

1. Review the discussion about round-off errors and the quadratic equation in the Session 2 notes with your partner. Make sure you go over the pseudo-code for the test case and understand it.
2. Look at the code quadratic\_equation\_1a.cpp in an editor. It represents the pseudo-code as it has just been typed in, mistakes and all. Note that it is not indented consistently. *Why might we care about formatting?*

**The desired output might be different**

1. Now try compiling with "make -f make\_quadratic\_equation\_1a". You should get a bunch of warnings and errors. Each one comes with the line number of the error. Take each one in turn and below what you think the error is (you'll need to identify the line numbers in your editor; ask an instructor if you don't know how to turn them on). Each warning is a clue to a mistake in the code (such as a typo)! [One hint: "setprecision" is defined in the header file "iomanip".] If you get stuck, quadratic\_equation\_1.cpp is the corrected version (without the "a" after the "1"). *List the errors here:*
   * **Iomanip was not present**
   * **“disc” did not have float in front of it**
   * **“disc” did not have float in front of it**
   * **The first “setprecision initially did not have endl;**
2. Correct these errors and see that it compiles and runs. Try a=1, b=2, c=0.1 and see that the roots differ. *Which two (of x1, x1p, x2, x2p) are the most accurate? Explain why the others are not as accurate.* (Hint: check the Session 2 notes.)

**X2 and X1P are the most accurate**

1. Now look at quadratic\_equation\_2.cpp, which is the second pass at the code. Check the comments for a description of changes. List below up to three C++ lines from the code that you don't understand completely (it may be most of them or it may be none!):  
     
   **I understand all of it**
2. Compile and run quadratic\_equation\_2.cpp, and enter a=1, b=2, c=.1 again. The output is now more detailed and an output file named "quadratic\_eq.dat" is created. We'll discuss outputting to a file more later; for now, this is an example you can follow to do it on your own (there were also examples from the Session 1 codes).
3. Follow the "Plotting Data from a File with Gnuplot" handout to duplicate the plot on the back. Try some of the extra commands, including output to a postscript file and opening it using an appropriate postscript viewer.
4. *Is the plot qualitatively and quantitatively consistent with the analysis in the notes? Explain in a few sentences. [Hints: What does a straight line mean? What does the slope tell you?]*

**The plot is plot qualitatively and quantitatively consistent with the analysis in the notes.**

**The straight line means that the graph somehow resembles a linear trend.**

**The slope tells me how much does the y value change after some amount of the x value.**

Summing in Different Orders (see Session 2 notes)

Here we consider another example of subtractive cancellation and how computer math is not the same as formal math. Imagine we need to sum the numbers 1/n from n=1 to n=N, where N is a large number. We're interested in whether it makes any difference whether we sum this as:  
   1 + 1/2 + 1/3 + ... + 1/N (summing up)  
or as:  
   1/N + 1/(N-1) + ... + 1/2 + 1 (summing down)  
In formal mathematics, of course, the answer are identical, but not on a computer!

To help think about the sum up vs. sum down problem, it's useful to think first about a simpler version. Suppose you want to numerically add 107 small numbers a = 5x10-8 to 1 (so the exact answer is 1.5). For the first three questions, *predict* the answer before running the code.

1. *If you add 1 + a + a + ... in single precision, what do you expect to get for the total?* (Hint: recall the discussion of "machine precision.")  
     
   **1 + a + a**
2. *If, instead, you add a + a + ... + 1 in single precision, do you expect a different answer? Which will be closer to the exact answer?*  
     
     
   **a + a + 1**

**a + a + 1 will be closer to the answer**

1. *If you repeat the exercise in double precision, what do you expect will happen?*  
     
   **the digits will add**
2. Write a brief "pseudocode" here for a program that would compare these two ways of summing. Then look at the file order\_of\_summation1a.cpp in an editor; *does it look like an implementation of your pseudocode?*
   * **Input float and int values**
   * **Declare a double variable**
   * **Do a for loop for adding the numbers**
   * **Use “set” syntax for precision**  
       
       
     **Yes the code implements my pseudocode**
3. In a terminal window, use the makefile make\_order\_of\_summation1a to try and compile it ("make -f make\_order\_of\_summation1a"). Identify and fix the errors, until it compiles and runs correctly (verify the results against the comments in the file). Correct your answers to 1,2,3 if necessary (and understand them!).
4. Run indent on the file ("indent order\_of\_summation1a.cpp") and check that it is now nicely formatted.
5. *Challenge: Predict the results for 108 additions (instead of 107):*

**The eps+eps+…+1 column will have larger values.**

**And I was right when I compiled the code.**  
  
  
  
  
Now change the code, and see if your prediction is correct (or that you can explain it after the fact).

Bessel 1: Making Another Plot with Gnuplot

The next three tasks (Bessel 1-3) build on the discussion of spherical Bessel functions in the Session 2 notes. Skim this discussion before going further.

1. Copy make\_area to make\_bessel and open make\_bessel in an editor. (If you are in the session\_02 subdirectory, you can use "cp ../session\_01/make\_area ." to create a new copy where you are or "cp ../session\_01/make\_area make\_bessel" to make a copy directly; do you understand what all the "."'s mean? If not, ask!) Follow the instructions in the makefile to convert it to make\_bessel. Run the makefile using "make -f make\_bessel".
2. If you run bessel.cpp, you'll generate a data file "bessel.dat". By looking at the code, figure out what the columns in the file mean. *What Bessel function is being output? What are the columns?*  
     
   **It outputs Spherical Bessel functions via up and down recursion. The first column is the x value being input to the Bessel function. The second column is the y value for the up recursion. The third column is the y value for the down recursion.**
3. Make a plot of the third column as a function of the first column using gnuplot. (Use the example in the handout as a guide. You might also find the "GNUPLOT Manual and Tutorial" useful.) Try plotting BOTH the second and third columns vs. the first column on the same plot (i.e., two curves). *Print out your plot.*

Chart, line chart, scatter chart

Description automatically generated