PHY 480

Session 09/10

Revisiting area.cpp with a C++ Class:

1. How does circle.h know to use circle.cpp? Is it because we include circle.cpp in the make file? The first circle is created with “Circle my_first_circle(my_radius)” outside the {}’s. The second circle is created with “Circle my_second_cirlce(2.*my_radius)” inside the {}’s since that’s where it was created. The second circle is destroyed inside the {}’s and the first circle is destroyed once we’re outside the {}’s since that’s where it was created. Thus, the second circle is destroyed before the first. The method “get_radius” allows us to obtain the radius of a “Circle” object while “set_radius” allows us to define the radius of a “Circle” object.

2. Each circle is destroyed at the end of the {}’s within which they were created. Since the second circle’s {}’s are inside the first circle’s {}’s, we reach the end of the second circle’s {}’s first and thus the second circle is destroyed first.

3. Yes it worked! (Just run test_Circle.x)

4. It doesn’t compile because you can’t access a private variable in the main function.

Optimization 101, Squaring a Number:

3. It took the pow(x,2) method 3.99358 seconds while it only took the x*x method 0.365803 seconds. Those the x*x method is faster than the pow(x,2) meth.

4. It took the squareit method 0.512112 seconds which is 0.133762 seconds (overhead) longer than the x*x’s method which took 0.37835 seconds but much faster than the pow(x,2) method which took 4.0384 seconds. I tried different values of repeat but it seems like the squareit method is never faster than the x*x method.

5. The macro method takes about the same amount of time as the x*x method.

6. The inline method takes about the same amount of time as the macro method and the x*x method.

7. Using -O2 reduced each method's time to run by a factor of 10^(-5). The fastest way is house the inline, macro, or x*x method with -O2 optimization.

GSL Differential Equation Solver:

4. See GSL_Differnetial_Equation_Solver_Plot. I see the plots gravitating around a central point.

GSL Interpolation Routines:

1. No questions.

3. See GSL_Interpolation_Routines_3

4. Yes

5. See GSL_Interpolation_Routines_5. The polynomial spline is best at the peak and worst outside the peak. The linear spline is best outside the spline and slightly off at the peak. The cubic spline is basically right on both at and outside the peak.

Command Line Mystery:

Not really

Python Scripts for C++ Programs:

1. Yes. Output:

radius = 5, area = 78.5398

radius = 10, area = 314.159

radius = 15, area = 706.858

radius = 20, area = 1256.64

radius = 25, area = 1963.5

2. Yes!

3. No questions:

Cubic Splining

3. See gsl_spline_test_class_hydrogen.cpp. With 500,000 points I got that only one point had an error of more than 0.000001. To check this I added an if statement that increased a counter overtime the absolute error in the derivative was greater than or equal to 0.000001.

4. See gsl_spline_test_class_hydrogen.cpp. 100,000 points is sufficient to get the integral using Riemann's method within 0.000001 of the exact integral.