Phys 512 Problem Set 1

Due on github Friday Sep 23 at 11:59 PM. You may discuss problems, but everyone must write their own code.

Problem 1: One can work out the electric field from an infinitessimally thin spherical shell of charge with radius R by working out the field from a ring along its central axis, and integrating those rings to form a spherical shell. Use both your integrator and scipy.integrate.quad to plot the electric field from the shell as a function of distance from the center of the sphere. Make sure the range of your plot covers regions with $z < R$ and $z > R$. Make sure one of your z values is R. Is there a singularity in the integral? Does quad care? Does your integrator? Note - if you get stuck setting up the problem, you may be able to find solutions to Griffiths problem 2.7, which sets up the integral.

Problem 2: Write a recursive variable step size integrator like the one we wrote in class that does **NOT** call $f(x)$ multiple times for the same x. The function prototype should be

```
def integrate_adaptive(fun,a,b,tol,extra=None):}
```

where extra contains the information a sub-call needs from preceeding calls. On the initial call, extra should be None, so the integrator knows it is starting off. For a few typical examples, how many function calls do you save vs. the lazy way we wrote it in class?

Problem 3: a) Write a function that models the log base 2 of x valid from 0.5 to 1 to an accuracy in the region better than 10^{-6}. Please use a truncated Chebyshev polynomial fit to do this - you can use np.polynomial.chebyshev.chebfit. How many terms do you need? You should use many x/y values and fit to some high order, then only keep the terms you think you'll need and drop the rest. Make sure also that you rescale the x-range you use to go from -1 to 1 before calling chebfit.

Once you have the Chebyshev expansion for 0.5 to 1, write a routine called mylog2 that will take the natural log of any positive number. Hint - you will want to use the routine np.frexp for this, which breaks up a floating point number into its mantissa and exponent. Relatedly, having your routine natively produce the log base 2 will simplify life. Also feel free to use np.polynomial.chebyshev.chebval to evaluate your fit. You might ask yourself if a computer ever takes a natural log directly, or if it goes through the log base 2 (near as I can tell, it's the log base 2).

If you'd like a bonus, repeat this exercise using the Legendre polynomial (or, heaven forbid, the actual Taylor series) of the same order as the Chebyshev you used. How big is the RMS error in the log compared to Chebyshev? How big is the maximum error? This hopefully gives you a flavor as to how useful Chebyshev's can be if you have to write your own function. If you enjoy this sort of problem, I'd encourage you to look at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.5177

that describes how some of these transcendental functions are actually implemented.