PHYS 512 – Problem Set 2

Jade Ducharme (260929684)

September 2021

Problem 1

For this problem I chose to use the Legendre polynomial integrator we saw in class. From Griffiths 2.7, we know the integral describing the electric field strength due to a thin spherical shell at a distance z from its center is given by:

$$E = \frac{kq}{2}\int_{-1}^{1}\frac{(z - Rx)dx}{(R^2 + z^2 - 2Rzx)^{\frac{3}{2}}}, \tag{1}$$

where $k \simeq 9 \times 10^9 \frac{\mathrm{Nm}^2}{\mathrm{C}^2}$ is the Coulomb constant, q is the total electric charge of the shell, z is the distance from the center of the shell, and R is the radius of the shell. I have randomly chosen values of $q = 0.5$ and $R = 2$ for the problem.

We also know that this integral has a closed form, given by:

$$E = \begin{cases} 0 & z \leq R \\ \frac{kq}{z^2} & z > R \end{cases} \tag{2}$$

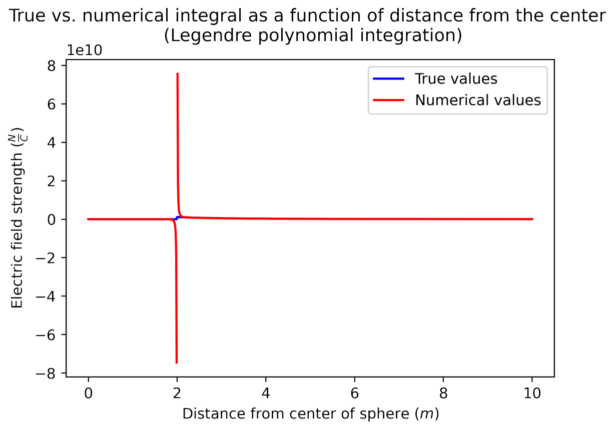
For my integrator, I choose values of z between 0 and 10 and plot the numerical result as well as the true result (given by Equation 2) as a function of z and present it in Figure 1.

Next, I did the same thing with scipy.integrate.quad and present the resulting plot in Figure 2.

There is a singularity in the integral given by Equation 1. It occurs when $R = z$ because at that moment the denominator is equal to zero when $x = 1$. My integrator doesn't know how to deal with it (as seen in Figure 1), but *scipy* has no trouble with it (as seen in Figure 2).

I also wanted to test the accuracy of my integrator vs. the *scipy.integrate.quad* integrator, so for each z value I computed the relative error using the following equation:

$$\text{err} = \frac{E_{true} - E_{numerical}}{E_{true} + 1} \tag{3}$$



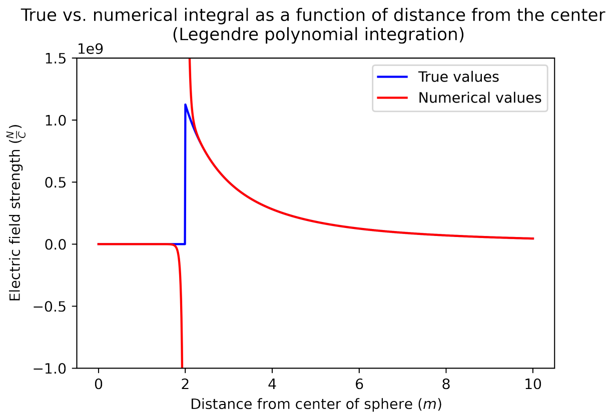


Figure 1: True integral vs. numerical integral as a function of the distance from the center of the thin spherical shell. (Left) Full breadth of y axis. (Right) Limits imposed on y axis to better observe the trend near $z = 2$.

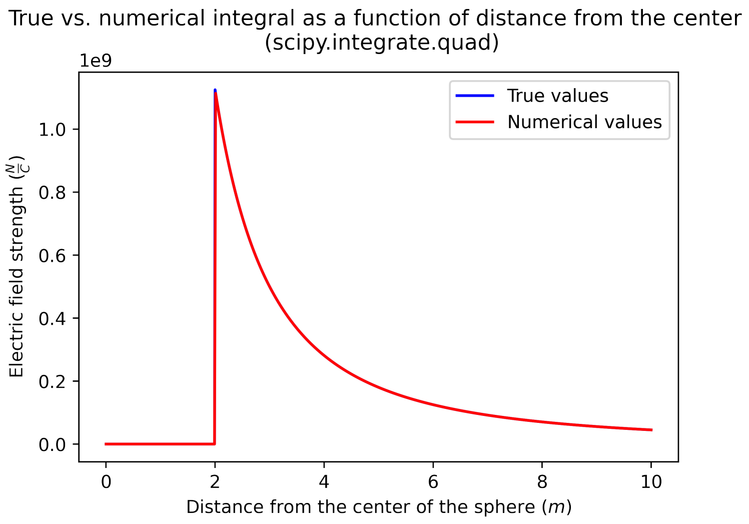


Figure 2: True integral vs. numerical integral computed using $scipy.integrate.quad$.

The reason for the $+1$ on the denominator is to avoid nan when $E_{true} = 0$ (which it is for all $z < R$). I plot the relative error as a function of z for my integral as well as for $scipy.integrate.quad$. I present both of these graphs in Figure 3.

Clearly, $scipy.integrate.quad$ performs better than my integrator for all values of z.

Problem 2

When we did it in class, each subdivision of the original x range would retain 3 out of 5 original data points. What I mean by this is say we originally pick five data points between 0 and 4. We obtain:

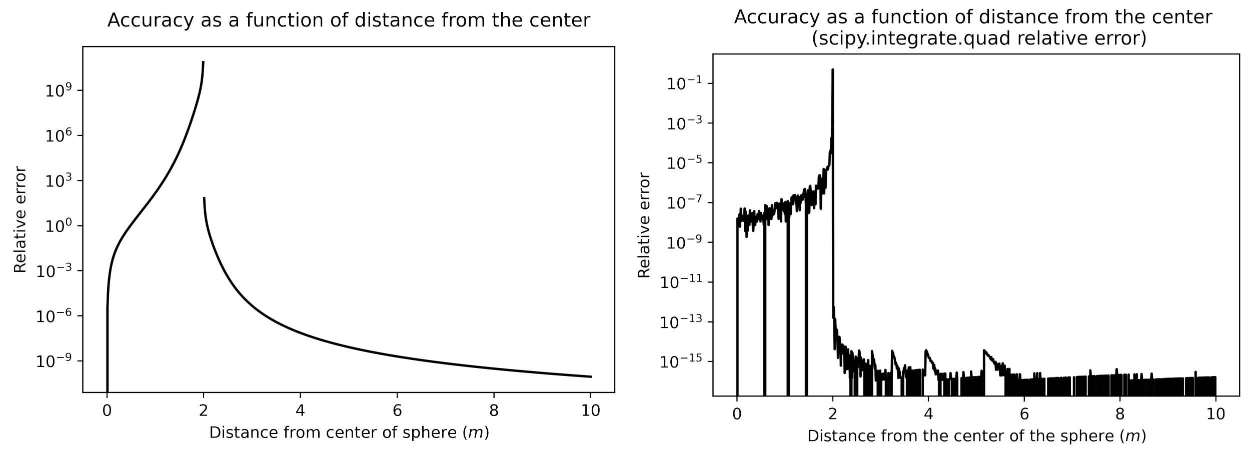


Figure 3: Relative error as a function of the distance from the center of the hollow shell for (left) my integrator and (right) *scipy.integrate.quad*. The y axis is on a log scale in both cases to better observe the trend.

$$x = [0, 1, 2, 3, 4]$$

If we slice this range in half and look at both halves individually, we obtain:

$$x_{left} = [0, 0.5, 1, 1.5, 2]$$

$$x_{right} = [2, 2.5, 3, 3.5, 4]$$

In both cases, the half-intervals contain 3 of the original interval points – x_{left}'s first, mid, and final point correspond to the first three points in x while x_{right}'s first, mid, and final points correspond to the last three points in x. Therefore, for our adaptive integrator, we can always save the first three points of the original range and distribute them to the left half, then save the last three points and distribute them to the right half.

Let's say an adaptive integrator needs to perform 10 splits in order to get an answer with the desired accuracy. The adaptive integrator we saw in class would therefore need to generate 5 data points 10 different times, yielding 50 function calls. On the other hand, my new-and-improved integrator generates 5 data points on the first call, but only needs to generate 2 additional points for each subsequent call, for a total of $5+9\times2 = 23$ function calls.

Let the number of splits be n, and let that be a very large number. For the 'lazy' integrator, we would get $5n$ function calls in total. For my improved integrator, I would get $5 + 2n \approx 2n$ function calls. The improved version only calls the function $\frac{2}{5}$ the amount of times the old version does – a good improvement!

Problem 3

In order to approximate the log base 2 of x between $\frac{1}{2}$ and 1, I first simulate a bunch of data points between $\frac{1}{2}$ and 1, take the $\log_2$ of those numbers, and run them through *np.polynomial.chebyshev.chebfit*. This gives my the Chebyshev coefficients. I then test this out using *np.polynomial.chebyshev.chebval* to make sure it actually works. In order to recover an accuracy in the region better than 10^{-6}, I found that I only need a Chebyshev expansion of degree 7 (so, 8 coefficients). This gives rise to Figure 4, where I plot the Chebyshev approximation for $\log_2(x)$ vs. the true value for $log_2(x)$ (found using *numpy.log2*).

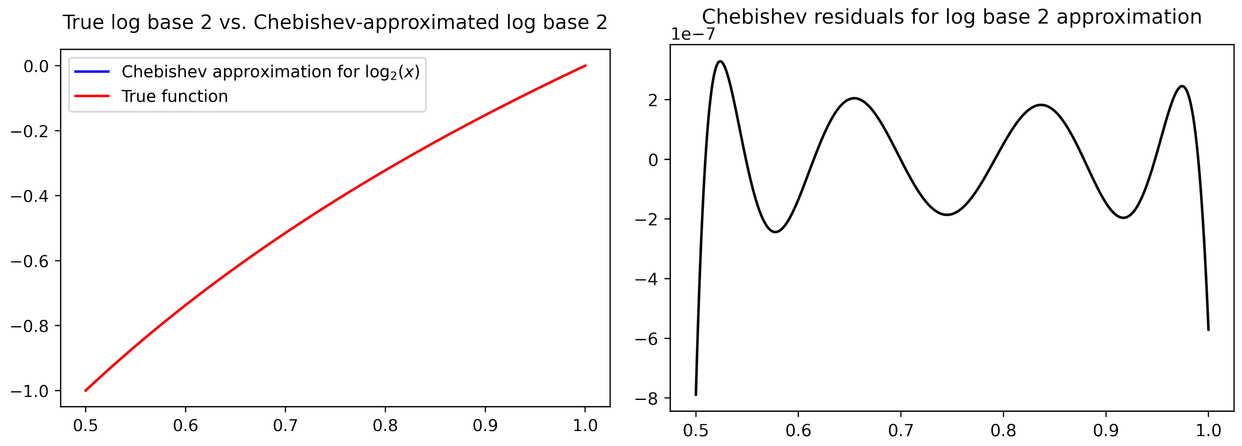


Figure 4: (Left) True $\log_2(x)$ vs. the Chebyshev degree 7 approximation for $\log_2(x)$. The two lines are indistinguishable by eye. (Right) The residuals for the left-hand plot show an accuracy greater than 10^{-6} throughout the region.

Next, if I want to approximate the natural logarithm of a given number, I can decompose that number into mantissa and twos exponent:

$$\begin{aligned} x &= a \times 2^b \\ \log_2(x) &= \log_2(a \times 2^b) \\ &= \log_2(a) + b \end{aligned} \tag{4}$$

Since the mantissa is always between -1 and 1, I can use my Chebishev approximation in order to approximate $\log_2(a)$ in Equation 4. Once I've successfully approximated the $\log_2$ of a given x, I can use the change-of-base log rule to turn it into a natural logarithm:

$$\begin{aligned} \ln(x) &= \frac{\log_2(x)}{\log_2(e)} \\ &= \frac{\log_2(a_x \times 2^{b_x})}{\log_2(a_e \times 2^{b_e})} \\ &= \frac{\log_2(a_x) + b_x}{\log_2(a_e) + b_e}, \end{aligned} \tag{5}$$

where in the second I've decomposed e into its own mantissa and twos exponent in order to be able to apply my Chebyshev approximation onto it. Using the above equations, I simulated some x data and approximated their natural logarithm, yielding the graphs presented in Figure 5.

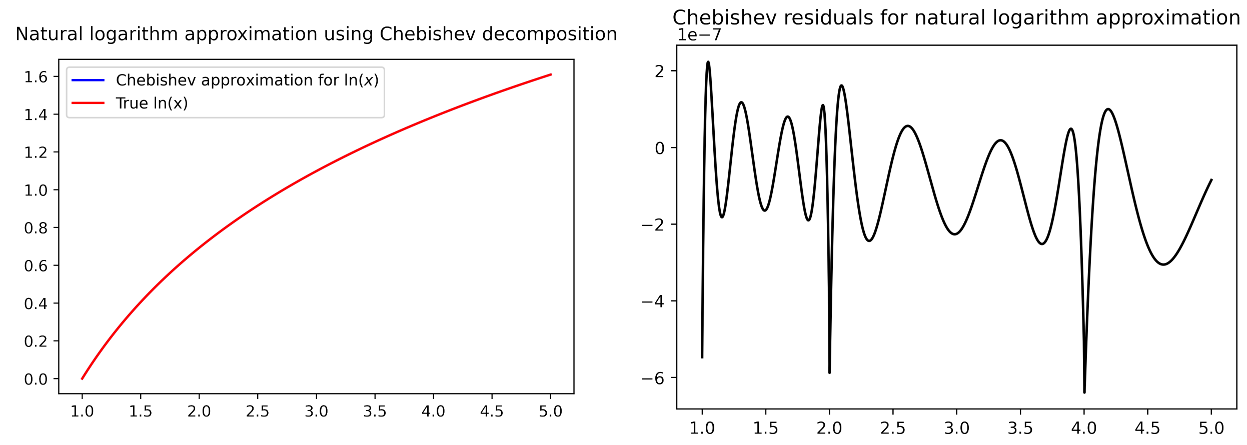


Figure 5: (Left) True $\ln(x)$ vs. the Chebyshev degree 7 approximation for $\ln(x)$. The two lines are indistinguishable by eye. (Right) The residuals for the left-hand plot show an accuracy greater than 10^{-6} throughout the region.