

PHYS 512 Assignment 2

Liam Fitzpatrick

September 2022

1 Question 1

This question was part of a problem set I completed last year in PHYS 350:

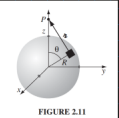


FIGURE 2.11

$$E = \frac{1}{4\pi\epsilon_0} \int \frac{\hat{r}}{r^2} \sigma dA$$

From the law of cosines:

$$r'^2 = z^2 + R^2 - 2Rz \cos\theta$$

E is only in z-direction, by symmetry x and y components cancel.

$$\hat{r}_z = \frac{(z - R \cos\theta) \hat{z}}{\sqrt{z^2 + R^2 - 2Rz \cos\theta}}$$

$$\Rightarrow E = \frac{\hat{z}}{4\pi\epsilon_0} \int_0^{2\pi} \int_0^\pi \frac{(z - R \cos\theta)}{(z^2 + R^2 - 2Rz \cos\theta)^{3/2}} \sigma R^2 \sin\theta d\theta d\phi$$

$$= \frac{\hat{z}}{4\pi\epsilon_0} \cdot 2\pi R^2 \int_0^\pi \frac{(z - R \cos\theta)}{(z^2 + R^2 - 2Rz \cos\theta)^{3/2}} \sin\theta d\theta$$

Let $u = -\cos\theta$
 $du = \sin\theta d\theta$
 $\theta = 0 \Rightarrow u = -1$
 $\theta = \pi \Rightarrow u = 1$

$$= \frac{\hat{z}}{2\epsilon_0} \int_{-1}^1 \frac{z + Ru}{(z^2 + R^2 + 2Rzu)^{3/2}} du$$

$$= \frac{\hat{z}}{2\epsilon_0} \int_{-1}^1 \frac{1}{\sqrt{3/2} \left(\frac{z^2 + R^2}{2z} + \frac{R^2}{2z} u \right)} \frac{du}{2Rz}$$

$$= \frac{\sigma R^2}{2\epsilon_0} \int_{-1}^1 \frac{1}{\sqrt{3/2} \left(\frac{z^2 + R^2}{2z} + \frac{R^2}{2z} u \right)} \frac{du}{2Rz}$$

Let $v = \frac{z^2 + R^2 + 2Rzu}{2z}$
 $dv = R du$
 $du = \frac{dv}{R}$
 $Ru = \frac{v - \frac{z^2 + R^2}{2}}{1}$

$$= \frac{\sigma R^2}{8z^2 \epsilon_0} \int \frac{z^2 + R^2 + 2Rz}{\sqrt{3/2} \left(\frac{z^2 + R^2}{2z} + \frac{R^2}{2z} u \right)} dv = \frac{\sigma R^2}{8z^2 \epsilon_0} \left[2(R^2 - z^2) v^{-1/2} + 2v^{1/2} \right]_{\frac{z^2 + R^2 - 2Rz}{2z}}^{\frac{z^2 + R^2 + 2Rz}{2z}}$$

$$= \frac{\sigma R^2}{4z^2 \epsilon_0} \left(\frac{R^2 - z^2}{\sqrt{z^2 + R^2 + 2Rz}} + \sqrt{z^2 + R^2 + 2Rz} - \frac{R^2 - z^2}{\sqrt{z^2 + R^2 - 2Rz}} - \sqrt{z^2 + R^2 - 2Rz} \right)$$

Case 1: $R > z$

$$E = \frac{\hat{z}}{4z^2 \epsilon_0} \left[\frac{(R-z)(R+z)}{z+R} + z+R - \frac{(R-z)(R+z)}{R-z} - (R-z) \right]$$

$$= \frac{\hat{z}}{4z^2 \epsilon_0} \left[\underbrace{R-z + z+R}_{=0} - R-z - R+z \right]$$

$$\vec{E} = 0$$

Case 2: $R < z$

$$E = \frac{\hat{z}}{4z^2 \epsilon_0} \left[\frac{(R-z)(R+z)}{z+R} + z+R + \frac{(z-R)(R+z)}{(z-R)} - (z-R) \right]$$

$$= \frac{\hat{z}}{4z^2 \epsilon_0} \left[2R + R + \frac{z^2 - R^2}{z-R} + z+R \right] = \frac{\sigma R^2}{z^2 \epsilon_0} \hat{z}$$

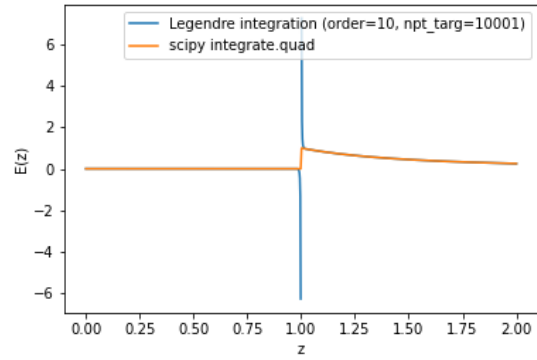
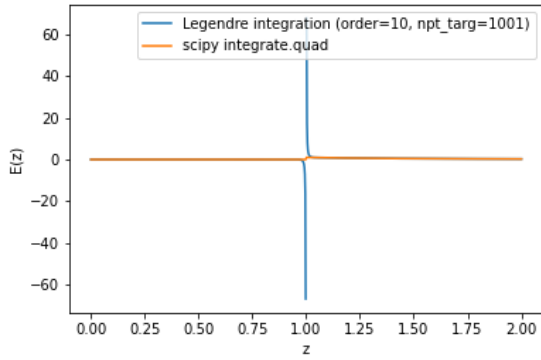
$$q = 4\pi R^2 \sigma \Rightarrow \sigma R^2 = \frac{q}{4\pi}$$

$$\Rightarrow \vec{E} = \frac{q}{4\pi z^2 \epsilon_0} \hat{z}$$

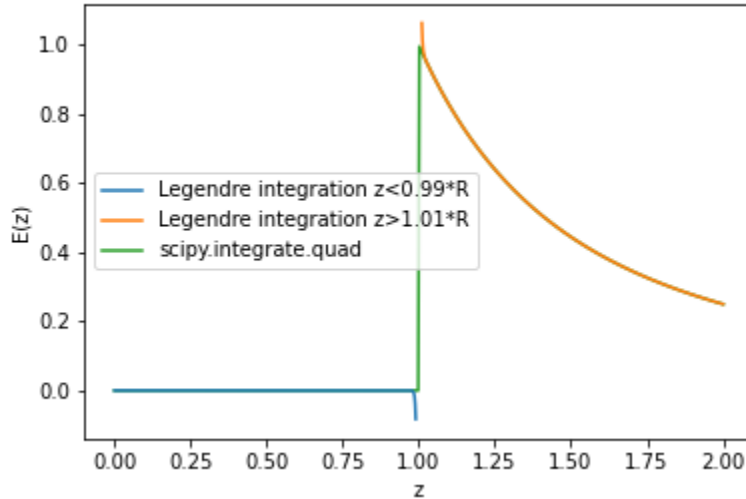
The denominator of the u-sub integral used for the numerical evaluation is 0 when,

$$z = R, u = -1 \Rightarrow z^2 + R^2 + 2Rzu = 0,$$

i.e. there is singularity at $z = R$. Using the Legendre integration method coded in class, it was first attempted to integrate the u-sub variable integral using order 10 polynomials and with 1001 points, then with 10001 points. Due to the pole from the denominator in the integrand, an infinite discontinuity was observed at $z=R$. Scipy integrate.quad did not care however about the discontinuity

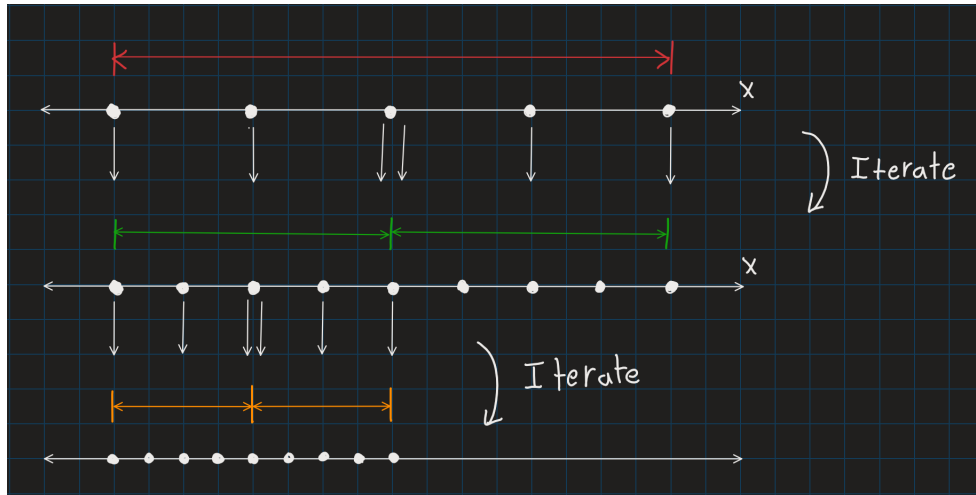


Using some physical intuition from the problem, a singularity in the integral is to be expected at $z = R$ because the charge density is infinite. Therefore we can remove a small region around the $z = R$ and instead evaluate an integral on each side, i.e. on the intervals $[0, R - \epsilon]$, $[R + \epsilon, \infty]$.



2 Question 2

For question 2, the same adaptive step size integrator was used from class, however it was modified by adding the 'extra' variable. To prevent calling a function again needlessly, function calls from previous integration iterations were stored in the extra variable. When the integration splits because the error is too large, the original domain (5 points) is now split into two 5 point regions by filling in extra points in between each of the original points. To illustrate this, here is a sketch:



In the sketch, the colored intervals show where the integration is being performed for each iteration. The arrows pointing down show how many function calls are saved by using the function calls from the previous iteration and which x points these correspond to. There are double arrows on the mid points to show that 2 function calls are saved, since the next iteration uses that point twice, once for each new integration interval. Therefore for each iteration, a total of 6 function calls are saved.

Here is the code for the adaptive integrator with a counter for the function calls saved from the initial code:

```
#Adding saved call count to adaptive integrate
def integrate_adaptive(fun,a,b,tol,extra=None):
    x=np.linspace(a,b,5)
    dx=x[1]-x[0]
    if extra==None: #Has no previous function calls stored
        calls_saved=0 #Initialize calls saved
        y=fun(x)
    else:
        calls_saved=extra[-1] #Saved calls count carried through in extra variable
        y=[extra[0],fun(x[1]),extra[1],fun(x[3]),extra[2]] #Mid and endpoints re-used from previous
        i1=(y[0]+4*y[2]+y[4])/3*(2*dx)
        i2=(y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])/3*dx
        myerr=np.abs(i1-i2)
        if myerr<tol:
            #print(calls_saved,'function calls saved on interval', [a,b]) #<-- use to see final integral intervals
            return np.array([i2, calls_saved])
        else:
            mid=(a+b)/2
            calls_saved+=6 #Each integration 'split' reuses 6 function calls of previous integration
            #Store mid and endpoints and updated saved call count for next iteration
            int1=integrate_adaptive(fun,a,mid,tol/2,extra=[y[0],y[1],y[2],calls_saved])
            int2=integrate_adaptive(fun,mid,b,tol/2,extra=[y[2],y[3],y[4],calls_saved])
            return int1+int2
```

Here is the output for a few examples:

```
Integral of sin(x) from 0 to pi/2: 1.000000001623166 with 888.0 function calls saved
Integral of 5x^4 from 0 to 1: 1.000000001552205 with 2304.0 function calls saved
Integral of ln(x) from 1 to e: 0.999999997950834 with 1134.0 function calls saved
```

3 Question 3

To model $\log_2(x)$, a function was written called `log2.chebfit(x_range,deg,pts)` which made use of numpy `chebyshev.chebfit` to fit Chebyshev polynomials of a certain degree to a given number of sampled points of

$\log_2(x)$ over some range of x values. The function returns the Chebyshev polynomial coefficients. Before the points were fitted however, the x range was scaled and shifted to $[-1,1]$.

```
#Fits Cheb. poly. with degree=deg given #points=pts over x_range
def log2_chebfit(x_range,deg,pts):
    x_pts=np.linspace(x_range[0],x_range[-1],pts)

    #Need to fit x in [-1,1]
    x_range=x_range*2/(x_range[-1]-x_range[0]) #Scaling interval length to 2
    x_range=x_range-x_range[0]-1 #Shifting interval to start at -1
    x_pts_new=np.linspace(x_range[0],x_range[-1],pts)

    log2_pts=np.log2(x_pts)
    cheb_fit=np.polynomial.chebyshev.chebfit(x_pts_new,log2_pts,deg)
    return cheb_fit
```

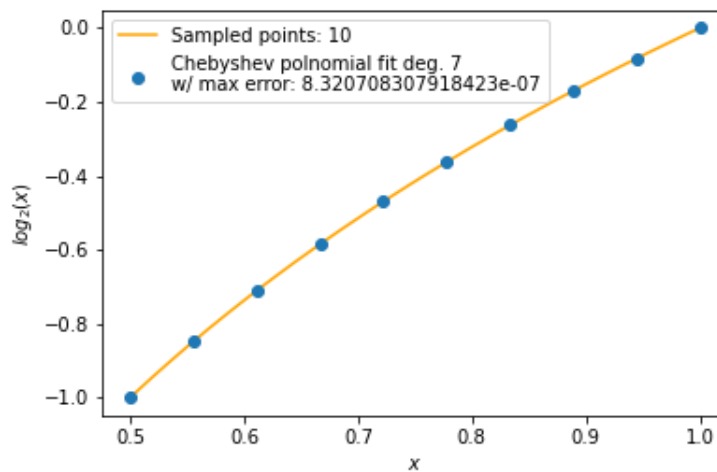
Next, a function called `log2_chebval(x,x_range,deg,pts)` was written which made use of `chebyshev.chebval` to evaluate the fit over some x range at a given x . The shifted x domain had to be taken into account when evaluating the polynomial.

```
#Evaluates cheb. poly. fit over x_range at a given x (not necessarily part of x_range)
def log2_chebval(x,x_range,deg,pts):
    cheb_fit=log2_chebfit(x_range,deg,pts)

    #Rescaling and shifting
    x_new=x*2/(x_range[-1]-x_range[0])
    x_range_new=x_range*2/(x_range[-1]-x_range[0])
    x_new=x_new-x_range_new[0]-1

    ans=np.polynomial.chebyshev.chebval(x_new,cheb_fit)
    err=np.abs(ans-np.log2(x))
    return ans, err, np.max(err)
```

Choosing an order 7 polynomial and fitting to 10 points over the x range $[0.5,1]$, this is the result with the maximum error less than 10^{-6} :



Using `np.frexp()`, we can split any float into its mantissa and exponent, where they are related by:

$$\text{float} = \text{mantissa} \times 2^{\text{exponent}}$$

$$\Rightarrow \log_2(\text{float}) = \log_2(\text{mantissa}) + \text{exponent}$$

Since the mantissa of a positive number is always between 0.5 and 1, we can use the previous result to find the natural log of any positive number. Here is the code written for the function `mylog2(x)` that performs this:

```
def mylog2(x):
    deg=7
    pts=10
    x_range=np.linspace(0.5,1,1001)

    mant_x,exp_x=np.frexp(x) #Evaluate log2(x) after splitting into mantissa + exp.
    log2_x=log2_chebval(mant_x,x_range,deg,pts)[0]+exp_x

    mant_e,exp_e=np.frexp(np.exp(1)) #Evaluate log2(e)
    log2_e=log2_chebval(mant_e,x_range,deg,pts)[0]+exp_e

    return log2_x/log2_e
```

To show that this works, here is a plot of `mylog2(x)` and `np.log(x)` from $x = 0.1$ to $x = 10$ and the plot of the difference between the two from $x = 10^{-3}$ to $x = 10^3$ (log scale):

