Problem 1:

As proposed, let's integrate the charged sphere with elements dq, contributing a field element $dE = \frac{dq}{4\pi\epsilon_0}\frac{1}{r^2}$.

↳ Recalling that $z^2 + R^2 - 2zR\cos\theta = r^2 \Rightarrow dE = \frac{dq}{4\pi\epsilon_0}\frac{1}{z^2+R^2-2zR\cos\theta}$

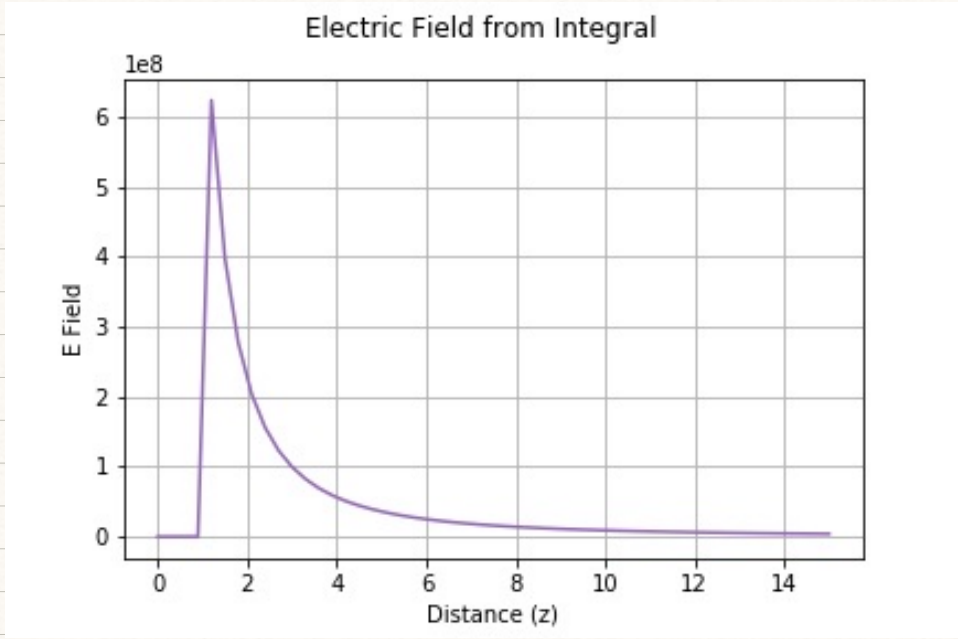
By Symmetry $\rightsquigarrow dE_z = \frac{dq}{4\pi\epsilon_0}\frac{z - R\cos(\theta)}{r(z^2-R^2-2zR\cos\theta)}$ where $\cos^{-1}\left(\frac{z-R\cos\theta}{r}\right)$ is the angle btwn $\hat{z}$ and dq...

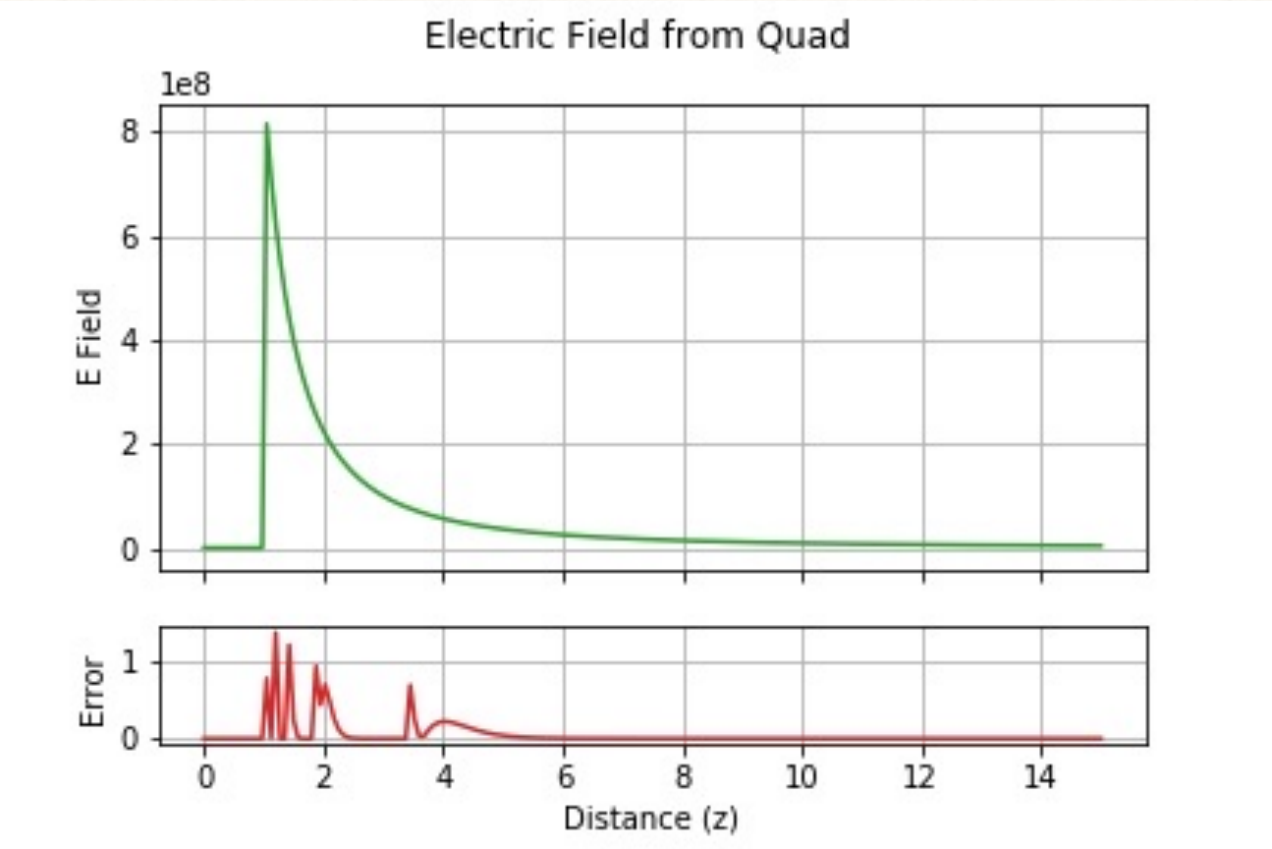
since $r = \sqrt{z^2+R^2-2zR\cos\theta}$ $\Rightarrow dE_z = \frac{dq}{4\pi\epsilon_0}\frac{z - R\cos\theta}{[z^2+R^2-2zR\cos\theta]^{3/2}}$

$\rightsquigarrow dE_z = \frac{[\sigma R^2 \sin\theta\, d\theta d\phi]}{4\pi\epsilon_0}\frac{z - R\cos\theta}{[z^2+R^2-2zR\cos\theta]^{3/2}}$

$\rightarrow E(z) = \frac{\sigma R^2}{2\epsilon_0}\int_0^{\pi}\frac{[z-R\cos\theta]}{[z^2+R^2-2zR\cos\theta]^{3/2}}\sin\theta\, d\theta$

Let's Integrate this numerically/graphically:





☆ Field is zero inside shell, spikes at R, then reduces at $\frac{1}{z^2}$.

↳ Integrator must skip z=R due to tolerance; however, quad can evaluate at R...

Problem 2:

```python
def integrate_adaptive(func, a, b, tol, extra=None):
    if extra == None:
        x = np.linspace(a,b,5)
        dx = (b - a)/(len(x) - 1)
        y = func(x)


        INT1 = 2*dx*(y[0]+4*y[2]+y[4])/3
        INT2 = dx*(y[0]+4*y[1]+2*y[2]+4*y[3]+y[4])/3
        absErr = np.abs(INT1 - INT2)

        integrate_adaptive.counter += 5 #Let's keep track of the number of integration iterations...

        if absErr < tol:
            return INT2


        else:
            middlePoint = (a + b)/2
            lowerBound = integrate_adaptive(func, a, middlePoint, tol/2, extra=[y[0], y[1], y[2], dx])
            upperBound = integrate_adaptive(func, middlePoint, b, tol/2, extra=[y[2], y[3], y[4], dx])
            return lowerBound + upperBound

    else:

        x=np.array([a+0.5*extra[3],b-0.5*extra[3]])
        y=func(x)
        integrate_adaptive.counter += 2
        dx=extra[3]/2
        area1 = 2*dx*(extra[0]+4*extra[1]+extra[2])/3
        area2 = dx*(extra[0]+4*y[0]+2*extra[1]+4*y[1]+extra[2])/3
        absErr = np.abs(area1-area2)

        if absErr < tol:
            return area2

        else:

            middlePoint = (a + b)/2
            lowerBound = integrate_adaptive(func, a, middlePoint, tol/2, extra=[extra[0],y[0],extra[1],dx])#a to mid ;; Quickly hinted in class as for the extras
            upperBound = integrate_adaptive(func,middlePoint,b,tol/2, extra=[extra[1],y[1],extra[2],dx])# mid to b
            return lowerBound + upperBound
```

```
Faster Integration Method:

 Lorentzian Function
Error:  2.2145707490039968e-10
Number of Calls: 2817

 Exponential Function
Error:  2.1464074961841106e-10
Number of calls: 6309

-------------------------------------

 Lazy Integration Method:

 Lorentzian Function
Error 2.2145707490039968e-10
Number of Calls: 7035

 Exponential Function
Error:  2.1464074961841106e-10
Number of Calls:  15765


=====================================

The count on the Lorentzian Integrator was improved by a factor of:  2.4973375931842385
The count on the Exponential Integration was improved by a factor of:  2.4988112220637184
This is roughly equivalent to our expected ratio of:  2.5
```

↝ Testing on Lorentzian and Exponential Functions...

↝ (5/2) Factor

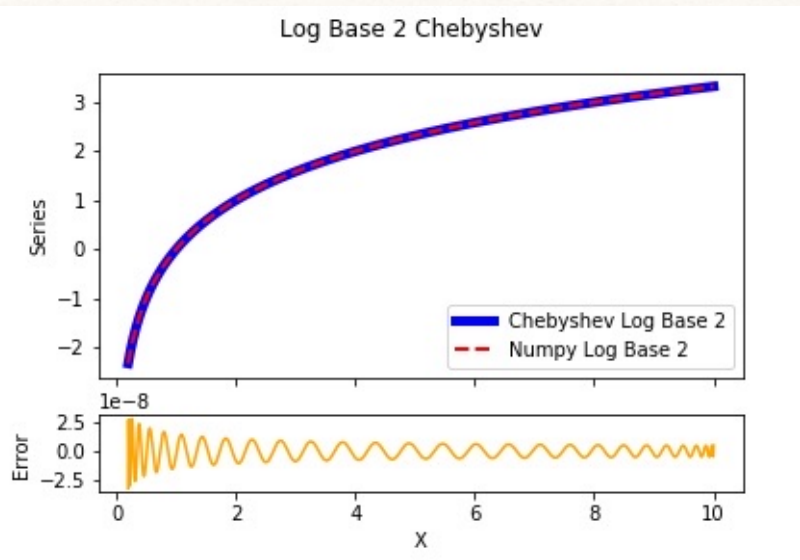
Problem 3:

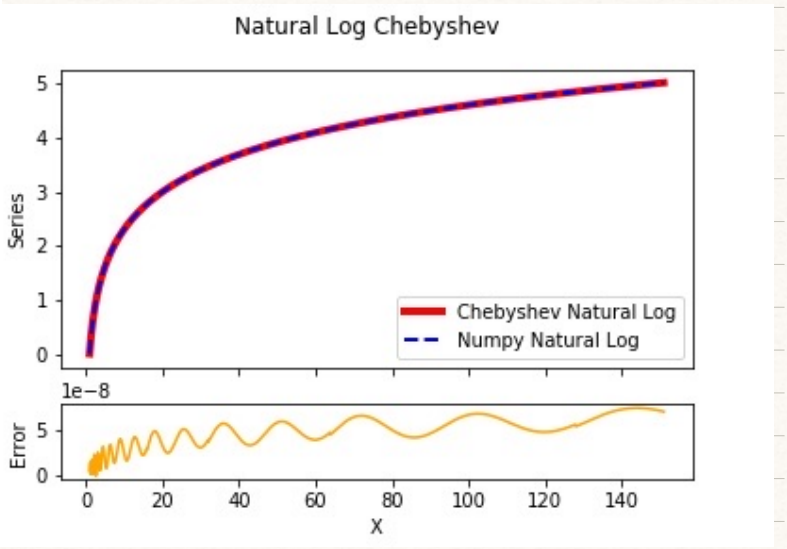
```
############################################## LOG 2 ##################################################
def chebyLog2(a= 1/5 , b = 10, ord = 200, tol = 1e-8):
    x = np.linspace(a, b, 2001) #Domain
    y = np.log2(x) #Log Base 2
    cheby = np.polynomial.chebyshev.Chebyshev.fit(x, y, deg=ord, domain = (a,b)) #Least Square
    cheby = cheby.trim(tol) #Trims all vals in cheby array below tol
    coeffCount = len(cheby)
    cheby_evl = cheby(x) #Polyn for every domain point from a to b
    print(cheby_evl, 'cheby Log 2')

    dataSet = np.vstack((x,cheby_evl)).T #vstack for full polyn evaluation (function)
    return dataSet, coeffCount, cheby

data,coeffCount,cheby = chebyLog2()
logError = np.log2(data[:,0])-data[:,1]
```

```
60
61 ############################################## NATURAL LOG ##################################################
62
63 def log2Func(num, tol = 1e-8, ErrVal = False):
64     mantis_num, exp_num = np.frexp(num)
65     mantis_e, expon = np.frexp(np.e)
66     print(mantis_num, exp_num, mantis_e, expon, 'log 2 cheb')
67     cheby = chebyLog2(tol = tol)[2]
68     log2_num = cheby(mantis_num) + exp_num
69     log2_e = cheby(mantis_e) + expon
70     natLog = log2_num / log2_e
71     logError = np.abs(np.log(num) - natLog)
72     print(logError, 'log error')
73     if ErrVal:
74         return natLog, logError
75     else:
76         return natLog
77
```





All code + notes provided in github files...