1. (a) We start by taking the Taylor expansions up to 3rd order of $f(x)$ evaluated δx and $2\delta x$ away:

$$f(x \pm \delta x) \approx f(x) \pm \delta x f'(x) + \frac{1}{2}\delta x^2 f''(x) \pm \frac{1}{6}\delta x^3 f'''(x) \tag{1}$$

$$f(x \pm 2\delta x) \approx f(x) \pm 2\delta x f'(x) + 2\delta x^2 f''(x) \pm \frac{4}{3}\delta x^3 f'''(x). \tag{2}$$

From eq.1 we can derive an initial numerical derivative:

$$f(x+\delta x) - f(x-\delta x) = 2\delta x f'(x) + \frac{1}{3}\delta x^3 f'''(x) \tag{3}$$

$$\implies f'(x) = \frac{f(x+\delta x) - f(x-\delta x)}{2\delta x} - \frac{1}{6}\delta x^2 f'''(x), \tag{4}$$

where we note the presence of an error term dependent on the third derivative of f.

Similarly for eq.2

$$f(x+2\delta x) - f(x-2\delta x) = 4\delta x f'(x) + \frac{8}{3}\delta x^3 f'''(x) \tag{5}$$

$$\implies f'(x) = \frac{f(x+2\delta x) - f(x-2\delta x)}{4\delta x} - \frac{2}{3}\delta x^2 f'''(x). \tag{6}$$

This similarly contains an error term dependent on the same order of δx and the third derivative of f.

We can now combine eq.4 and eq.6 to obtain a numerical derivative no error term (at this order...)

$$f'(x) - \frac{1}{4}f'(x) = \frac{f(x+\delta x) - f(x-\delta x)}{2\delta x} - \frac{1}{6}\delta x^2 f'''(x) \tag{7}$$

$$- \frac{1}{4}\left(\frac{f(x+2\delta x) - f(x-2\delta x)}{4\delta x} - \frac{2}{3}\delta x^2 f'''(x)\right) \tag{8}$$

$$\frac{3}{4}f'(x) = \left(f(x+\delta x) - f(x-\delta x) - \frac{f(x+2\delta x) - f(x-2\delta x)}{8}\right)\frac{1}{2\delta x} \tag{9}$$

$$\therefore f'(x) = \left(f(x+\delta x) - f(x-\delta x) - \frac{f(x+2\delta x) - f(x-2\delta x)}{8}\right)\frac{2}{3\delta x} \tag{10}$$

(b) To determine the numerical error on this operator, we need to include the 5th order terms of the taylor expansions:

$$f(x \pm \delta x) \approx ... \pm \frac{1}{120}\delta x^5 f^{(5)}(x) \tag{11}$$

$$f(x \pm 2\delta x) \approx ... \pm \frac{4}{15}\delta x^5 f^{(5)}(x). \tag{12}$$

replacing each term in eq.10 by their respective errors we get

$$err_{num}(f'(x)) = \left| \left(\frac{1}{60}\delta x^5 f^{(5)}(x) - \frac{1}{15}\delta x^5 f^{(5)}(x) \right) \frac{2}{3\delta x} \right| = \frac{1}{30}\delta x^4 f^{(5)}(x). \quad (13)$$

We can then plug this formula into the method we used to calculate the optimal dx:

$$err_{tot}(f'(x)) = \frac{f(x)\cdot(g\epsilon)}{\delta x} + err_{num}(f'(x)) \quad (14)$$

$$= \frac{f(x)\cdot(g\epsilon)}{dx} + \frac{1}{30}\delta x^4 f^{(5)}(x) \quad (15)$$

$$\approx \frac{f(x)\cdot\epsilon}{dx} + \frac{1}{30}\delta x^4 f^{(5)}(x). \quad (16)$$

This error can then be minimized:

$$\frac{\mathrm{d}err_{tot}(f'(x))}{\mathrm{d}\delta x} = 0 \quad (17)$$

$$-\frac{f(x)\cdot\epsilon}{\delta x^2} + \frac{4}{30}\delta x^3 f^{(5)}(x) = 0 \quad (18)$$

$$\implies \delta x = \left(\frac{30}{4}\frac{f(x)\epsilon}{f^{(5)}(x)} \right)^{\frac{1}{5}} \quad (19)$$

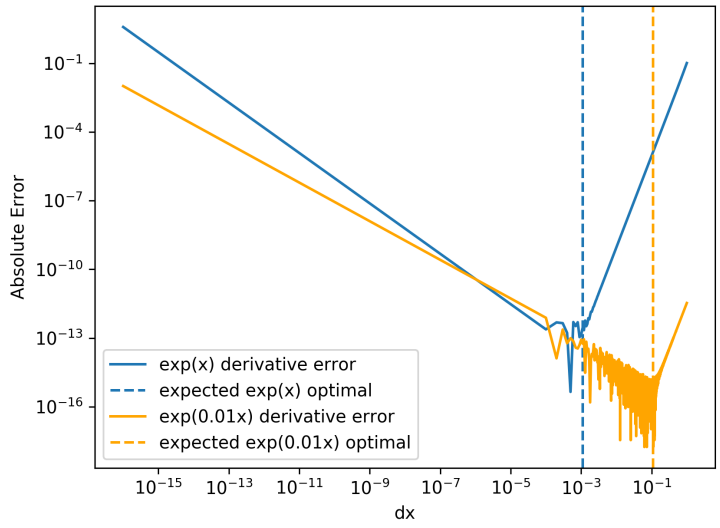

Figure 1: Error in the derivative for differing dx values for the given functions $exp(x)$ and $exp(0.01x)$. Dashed lines indicate the theoretically expected optimal values of dx calculated with the above formula. See derivative.py for generation script.

2. For the error analysis of this question, wikipedia states that the error on a nth order polynomial interpolation of a function f is given as

$$f(x) - p_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

for some point c in the interval of the points x_i used for interpolation. Thus I opted to compute this value by using numpy to take 4 numerical derivatives of the given data, and use the largest value of the three points used in the interpolation to give an upper bound. This resuled in an error of basically always zero, so I opted for an other method.

Since we use 4 points to do a cubic interpolation, by shifting the sets of points we use by one to the left or right we still obtain an interval that contains the x value we wish to interpolate at. So by taking the max of the difference between the interpolated value from the original 4 points and that of the shifted sets, we can get a bound on the error.

3. On the first run, we need to evaluate the function at 5 points $\{x_1, x_2, x_3, x_4, x_5\}$. When we split this interval into two, the left interval contains $\{x_1, x_2, x_3\}$ as it's begining, midpoint and endpoint respectively. Similarly the right interval has $\{x_3, x_4, x_5\}$ occupying the same roles. For each of these sub intervals, we thus need only to evaluate the function at an additional two points $\{x_1', x_2'\}$ lying at the points inbetween each intervals midpoint and endpoints.

Therefore, this method requires $5 + 2(N - 1)$ total function evaluations, where N is the number of times we the method is called. The old method required $5N$ function evaluations. Since I made no changes to the tolerance or precision of the new method, N should be the same for all cases. So for 'large' N we expected the new method to use roughly 2/5 as many function calls.

The provided script compares the two methods for integrating $\frac{1}{1+x^2}$ and uses 69 function calls in the new method vs 165 for the old one. Giving a ratio of $0.4\bar{18} \approx \frac{2}{5} = 0.4$.

4. There's a singularity that occurs when your point of interest is $z = \pm R$. scipy.integrate.quad does not care about this point, however, my lazily programmed code does. Even worse, as the integration approaches this point, the number of iterations needed blows up and reaches the recursion depth. I got arround this by setting the value of the integral to be infinite whenever the recursion depth ws reached using a simple try/except statement in python...

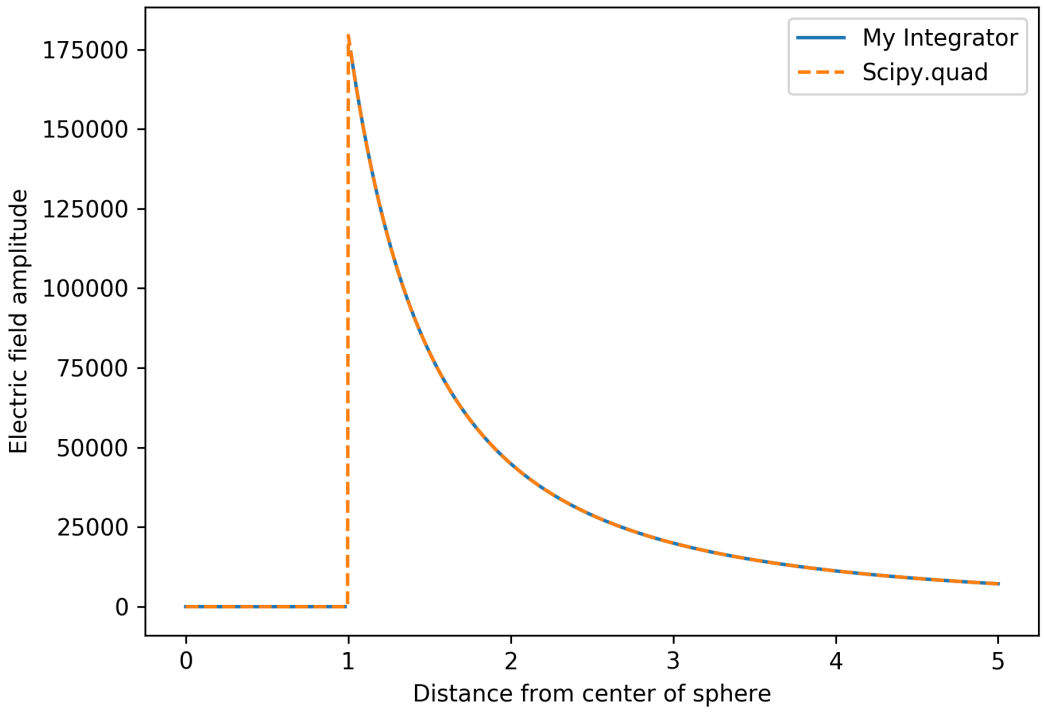

Figure 2: Electric field of a spherical shell with radius 1 m and charge 10E-6 Q. Two integration methods are used, my variable integrator, and scipy.quad.