PHYS 512 - Computation Physics with Applications

Nicola Grenon Vinci

1. (a) Taylor series expansions:
$f(x+dx) = f(x) + f'(x)dx + \frac{1}{2!}f''(x)dx^2 + \frac{1}{3!}f'''(x)dx^3 + \frac{1}{4!}f^{(x)}(x)dx^4 + O(dx^5)$
$f(x-dx) = f(x) - f'(x)dx + \frac{1}{2!}f''(x)dx^2 - \frac{1}{3!}f'''(x)dx^3 + \frac{1}{4!}f^{(x)}(x)dx^4 + O(dx^5)$
$f(x+2dx) = f(x) + 2f'(x)dx + \frac{2^2}{2!}f''(x)dx^2 + \frac{2^3}{3!}f'''(x)dx^3 + \frac{2^4}{4!}f^{(x)}(x)dx^4 + O(dx^5)$
$f(x-2dx) = f(x) - 2f'(x)dx + \frac{2^2}{2!}f''(x)dx^2 - \frac{2^3}{3!}f'''(x)dx^3 + \frac{2^4}{4!}f^{(x)}(x)dx^4 + O(dx^5)$

We can take the symmetrized approximation with two points and reduce the the error by one order of dx.
$f(x+dx) - f(x-dx) = 2f'(x) + \frac{1}{3}f'''(x)dx^3 + \frac{1}{60}f^{(5)}(x)dx^5 + O(dx^7)$
$f(x+2dx) - f(x-2dx) = 4f'(x) + \frac{8}{3}f'''(x)dx^3 + \frac{8}{15}f^{(5)}(x)dx^5 + O(dx^7)$

Finally, combining the approximation from four points we can reduce the error in $f'(x)$ to order $O(dx^4)$.
$f'(x) = \dfrac{8[f(x+dx) - f(x-dx)] - [f(x+2dx) - f(x-2dx)]}{12dx}$

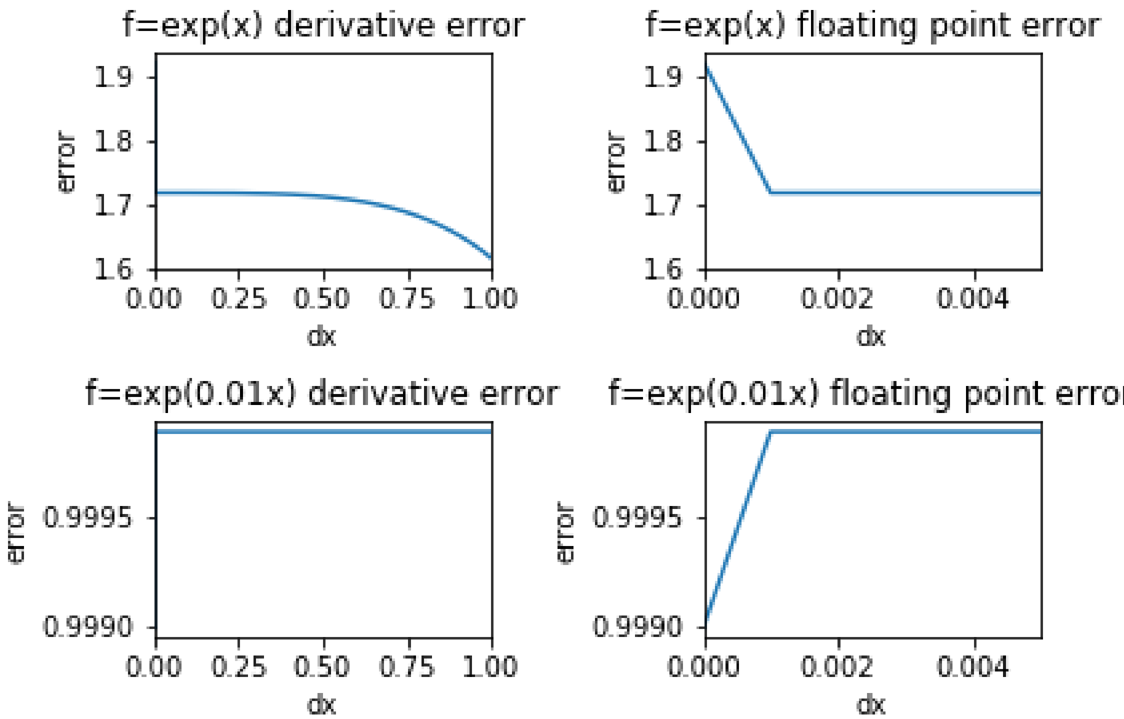
```python
def fourptsderiv(x, dx, fun):
    return (8*(fun(x+dx)-fun(x-dx))-(fun(x+2*dx)-fun(x-2*dx)))/(12*dx) #
```

(b) The truncation error in $f'(x)$ using a four points taylor approximations is $e_t = -\frac{1}{30}dx^4 + O(dx^5 f^{(4)})$. The floating point error is operations of addition or substraction is $e_r \approx 10^{-16}$ for double precision values. With the truncation error e_t and the roundoff error e_r, the derivative reads
$f'(x) = \dfrac{8([f(x+dx)+\epsilon_1] - f(x-dx)+\epsilon_2) + [(f(x+2x)+\epsilon_3) - (f(x-dx)+\epsilon_4)]}{12dx} + O(dx^4 f^{(4)})$
$= \dfrac{8[f(x+dx) - f(x-dx)] - [f(x+2dx) - f(x-2dx)]}{12dx} + O(dx^4 f^{(4)}) + \dfrac{(8\epsilon_1 - \epsilon_2) - (\epsilon_3 - \epsilon_4)}{12}$
Since the float point error $\dfrac{(8\epsilon_1 - \epsilon_2) - (\epsilon_3 - \epsilon_4)}{12dx} \leq \dfrac{3\epsilon_r}{2dx}$ and $|O(dx^4 f^{(4)})| \leq \frac{1}{30}dx^4 f^{(4)}(x)$,
$|e| = e_t + e_r \leq -\frac{1}{30}dx^4 M^{(4)}(f(x)) + \frac{3}{2}e_r$

We want to minimize the error:
$\frac{de}{dx} = 0 = -\frac{2}{15}dx^3 M^4 - \frac{3}{2}\frac{e_r}{dx^2}$
We put $e_r \approx 10^{-16}$ and we get

$$dx_{opt} \approx (\frac{5e_r}{M^4})^{1/5}$$

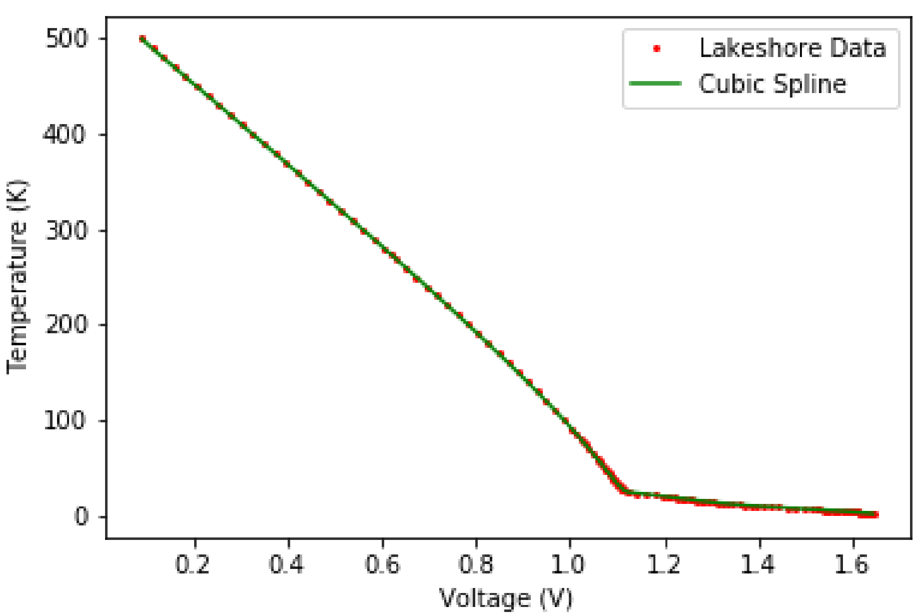
If we evaluate $f(x) = exp(x)$ at $x = 1$, let $M = e$ the optimal dx is $dx \sim 6.2 \cdot 10^{-4}$.
If we evaluate $f(x) = exp(0.01x)$ at $x = 1$, let $M = e$ the optimal dx is $dx \sim 1.4 \cdot 10^{-3}$. The following graphics show the error in the derivative for different dx compared to the true value as well as a zoom-in near the floating point error.



We can see the floating point error occuring for $f(x) = exp(x)$ and $f(x) = exp(0.01x)$ at $dxx \sim 0.001$.

2. The lakeshore data on diodes can be interpolated using the method interp1d to 3rd order from scipy.interpolate.

```
# Neighbours cubic spline
cubic_spline = interpolate.interp1d(diodes[1], diodes[0], kind='cubic')
x_spline = np.linspace(diodes[1,0],diodes[1,len(diodes[1])-1],1000)
y_spline = cubic_spline(x_spline)
```

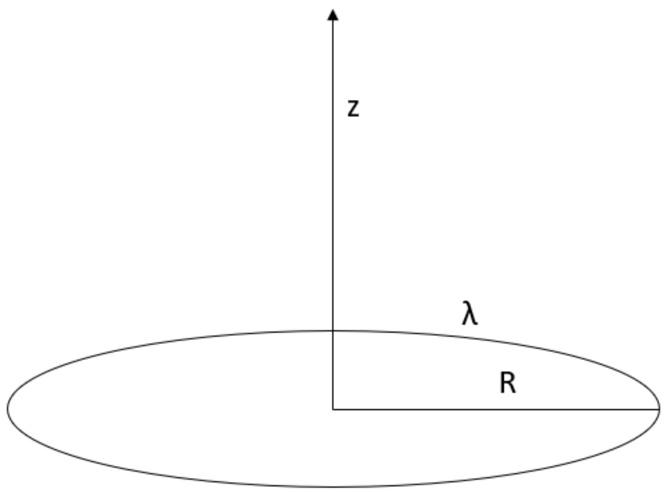


3. The electric field at a distance z above the center of a charged of ring of radius R and charge density

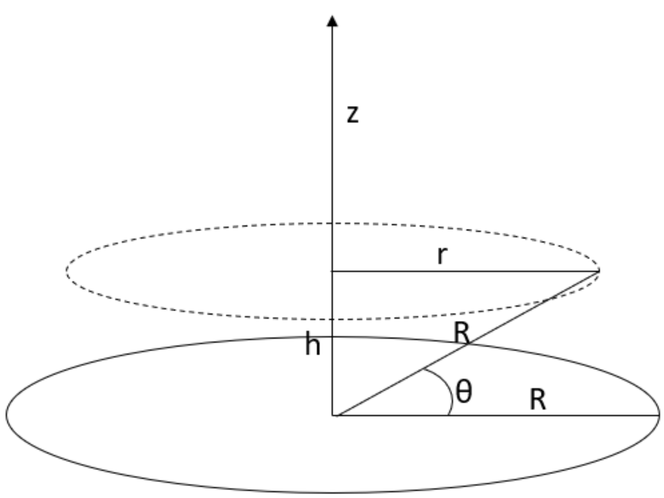
λ is given by

$$E_ring = \frac{\lambda}{2\epsilon_0} \frac{rz}{(z^2 + r^2)^{3/2}}$$

with the permittivity constant $\epsilon_0 \approx 8.854 F \cdot m^{-1}$.

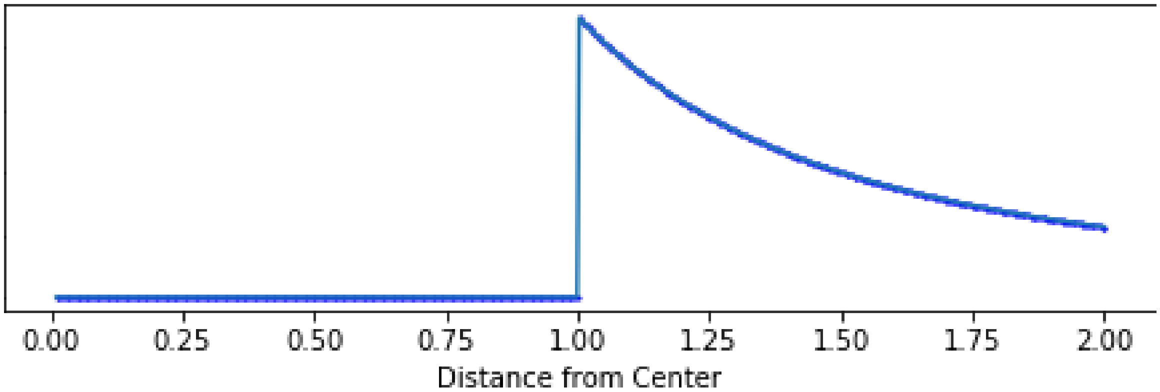


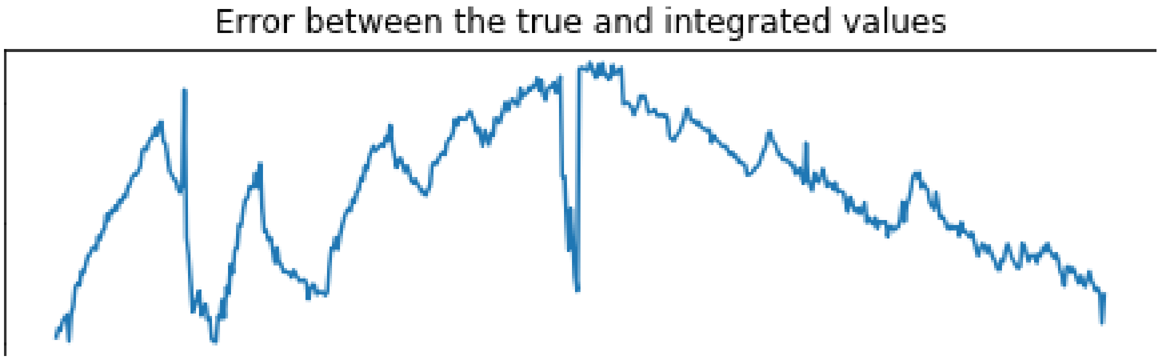
We can integrate the electric field of the ring over the angle of the sphere to get an integral to evaluate numerically for the electric field of the sphere:
$h \to Rsin(\theta)$
$r \to \sqrt{R^2 - h^2} = Rcos(\theta)$
$z \to z - h = Rsin(\theta)$
$\lambda \to \sigma = \frac{Q}{4\pi R^2}$



$$E_{sphere} = \frac{\sigma R}{2\epsilon_0} \int_{-\pi/2}^{\pi/2} \frac{cos(\theta)(z - Rsin(\theta))d\theta}{[(z - Rsin(\theta))^2 + (Rcos(\theta))^2)]^{3/2}}$$

Using the variable step size integrator from the last problem to integrate the integral above at different values of z, we get the following plot:





[h]

At the value $z = R$, we get a singularity where the integral approaches 0 from $z < R$ and ∞ from $z > R$. We avoided using the value $z = R$ or a value to close to it since the number of recursive steps quickly explodes. If instead we use the integrate.quad from the scipy library, we are able to compute a value for $z = R$, which comes in-between its two neighbours, but that value is wrong due to float point error.