PHY407 Lab 2

Teophile Lemay 1005036435

Charlie Hughes 1005231018

Question 1 (Charlie)

1 (a).

```
# array = numpy.loadtxt(filename)
# std_true = np.std(array, ddof=1)
# mean = np.mean(array)
# n = np.size(array)
# std1 = ( (1/(n-1)) * np.sum( (array – mean)**2) ) ** 0.5
# std2 = ( (1/(n-1) * (np.sum(array**2) – n*mean**2) ) ** 0.5
# print (std1 – std_true) / std_true
# print (std2 – std_true) / std_true
```

1 (b)

Relative error using eq. 1: -1.75e-16

Relative error using eq. 2: 3.74e-09

Clearly the relative error of eq. 2 is larger than eq 1.

1 (c)

Sequence 1:

Eqn 1 = -4.49e-16

Eqn 2 = 4.49e-16

Sequence 2:

Eqn 1 = -1.0e-15

Eqn 2 = -0.087

The difference between the two outputs is fairly minimal in sequence 1, but the one pass method performs far worse in sequence two (where the mean is quite high). Evidently an array with very large values and small differences (i.e. a small true standard deviation) is problematic. This is likely because if the elements are quite large (as they are in sequence 2 ~10^7), the differences between them are relatively small and calculating formula 2 requires extracting a small difference from two large numbers – leaving you vulnerable to Python's rounding.

1 (d)

Eqn 2 = -0.087

Eqn 2 (corrected) = -0.079

To fix this, the solution may be to rearrange the equation in some way that the equation $\sum x^2 - n\overline{x^2} \sim 0$ doesn't appear in the code (the difference is so close to 0 it is hitting Python's decimal limits). By scaling up x^2 and $n\overline{x^2}$ by many orders of magnitude, the difference between them becomes bigger (and less prone to error). By applying this workaround, the error was reduced somewhat (around 10%), still far worse than the two-pass system.

Question 2 (Teophile)

2 (a)

i.

$$I(0,1) = \int_0^1 \frac{4}{1+x^2}\,dx$$
$$= 4\int_0^1 \frac{1}{1+x^2}\,dx$$
$$4[arctan(x)]_0^1$$
$$= 4[arctan(1) - arctan(0)]$$
$$= \pi$$

ii. N = 4 slices
h = (1 - 0)/4 = ¼
First: trapezoidal rule
Using the equation for the trapezoidal rule from Lecture 2 notes, substituting $f(x) = \frac{4}{1+x^2}$, a = 0, b = 1, N = 4 and h = ¼ , we have:

$$I(0,1) \approx \frac{1}{4}\left[\frac{1}{2}\frac{4}{1+0^2} + \frac{1}{2}\frac{4}{1+1^2} + \sum_{k=1}^{3}\frac{4}{1+(0+k/4)^2}\right] = 3.131176471$$

Next: Simpson's rule
Using the equation for Simpson's rule from Lecture 2 notes, using the same substitutions:

$$I(0,1) \approx \frac{1}{4*3}\left[\frac{4}{1+0^2} + \frac{4}{1+1^2} + 4\sum_{k=1,3}\frac{4}{1+(0+k/4)^2} + 2\sum_{k=2}\frac{4}{1+(0+k/4)^2}\right]$$
$$= 3.141568627$$

iii. Code found in "lab2_Q02a.py"
Using the trapezoidal rule, an error on the order of 10^{-9} (-9.934109534981417e-09) was achieved using $N=2^{12}$ bins. Simpson's rule performed much better, achieving the desired error threshold (exact error -2.364971329882337e-09) with only $N=2^4$ bins. The functions were both timed using the built-in `timeit.timeit` function over 1000 repetitions and the mean times were recorded. The trapezoidal rule function took an average of $1.7 \cdot 10^{-3}$ seconds to run, while the Simpson's rule function took an average of $8.4 \cdot 10^{-6}$ seconds. Since the integration functions used have similar computational complexity, roughly O(N) (in big O notation), and contain similar operations, the time difference between the functions is likely due to the much larger number of bins required by the trapezoidal rule to achieve an error on the order of 10^{-9}.
Printed outputs below show proof of timing:

```
>>> timeit.timeit('int_trapezoidal(0, 1, my_func, 2**12)', setup='from lab2_Q02a import int_trapezoidal, my_func',  number=1000)/1000
0.0017474125999999614
```

```
>>> timeit.timeit('int_simpson(0, 1, my_func, 2**4)', setup='from lab2_Q02a import int_simpson, my_func',  number=1000)/1000
8.411400000113645e-06
```

iv. Code found in "lab2_Q02a.py"
The approximate error of the trapezoidal rule integrating $f(x) = \frac{4}{1+x^2}$ from 0 to 1 using 32 bins is $1.6 \cdot 10^{-4}$

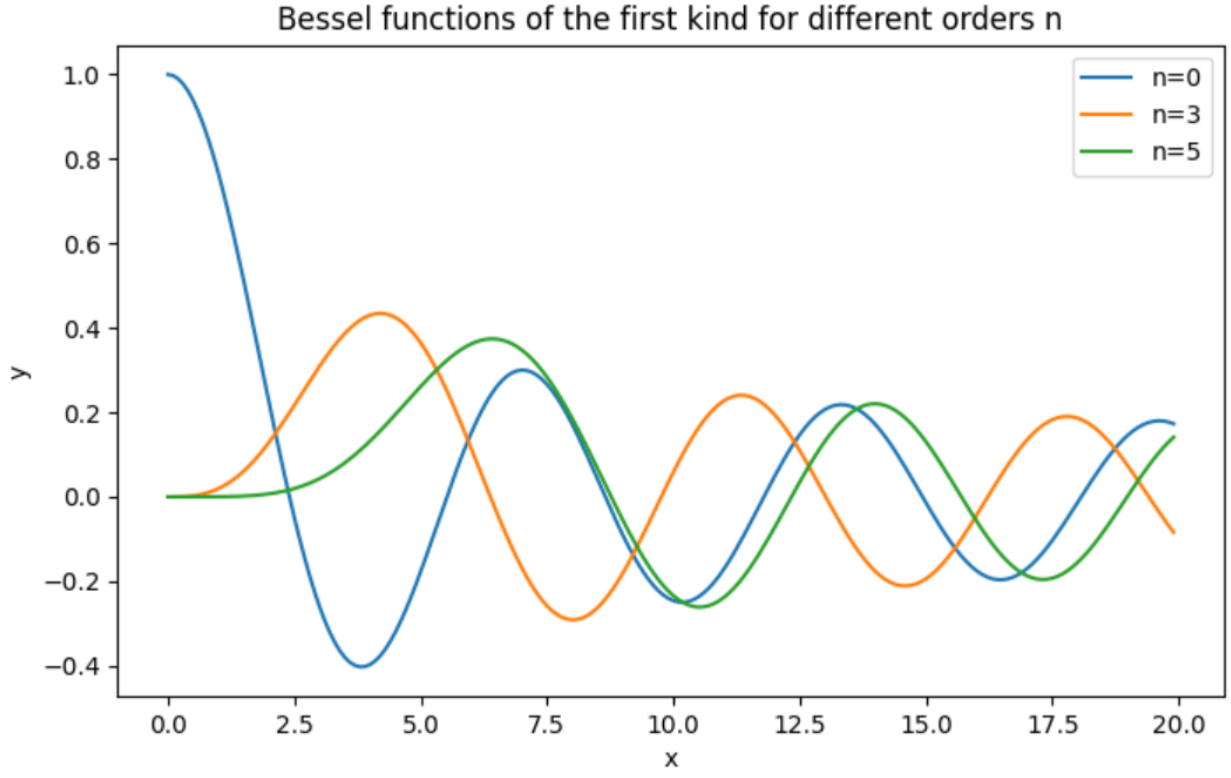
v. The practical estimation of errors method for trapezoidal rule integration would not be an accurate estimate of error for Simpson's method because the trapezoid rule error is on the order of h^2 while the error for Simpson's rule is on the order of h^4. This means that doubling N decreases the error by a factor of 16 rather than 2. To adapt the practical estimation of errors, we would need to divide the difference between the integrals by 15 instead of by 3.

2 (b).

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} cos(n\phi - x\,sin\,\phi)\,d\,\phi$$

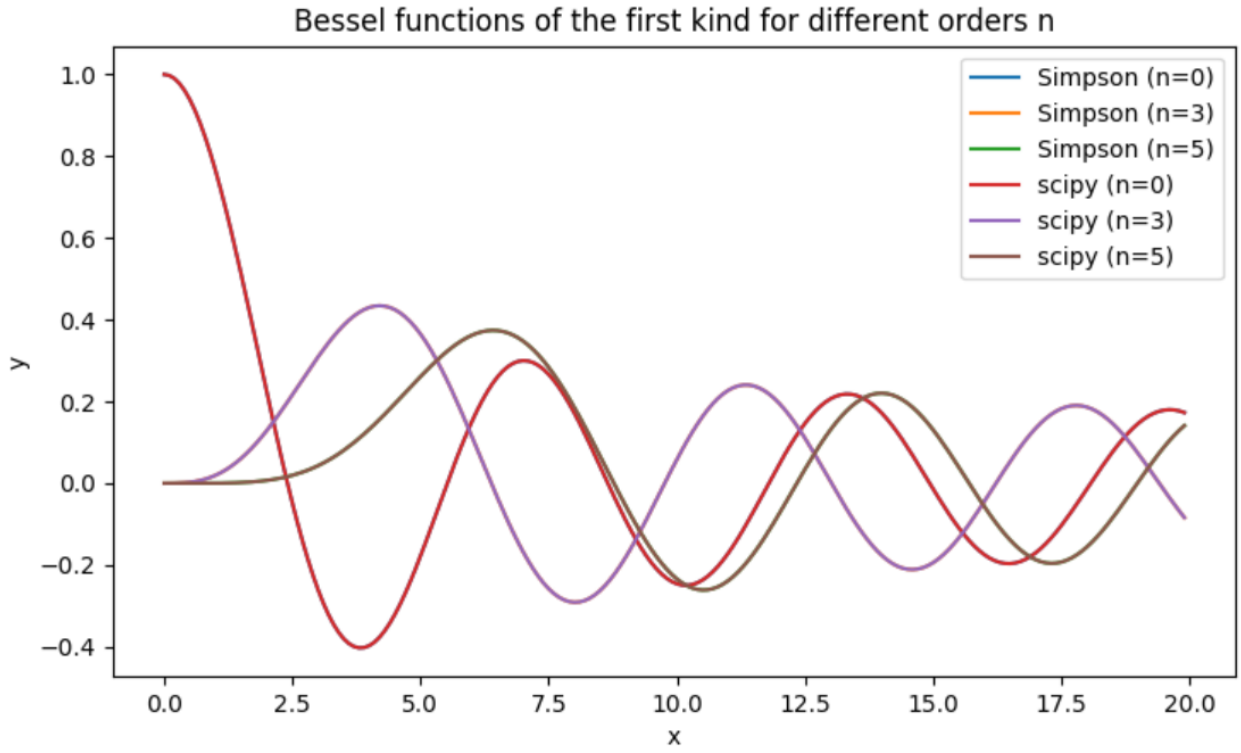
This integral is calculated using Simpson's rule with N=1000 bins for values of x from 0 to 20.

Figure: Bessel functions of the first kind, calculated using Simpson's rule.



The graphs of the Simpson's rule approximated Bessel functions of the first kind were compared to those calculated using `scipy.special.jv`.

Figure: Bessel functions of the first kind calculated with Simpson's rule compared to those calculated with `scipy.special.jv`

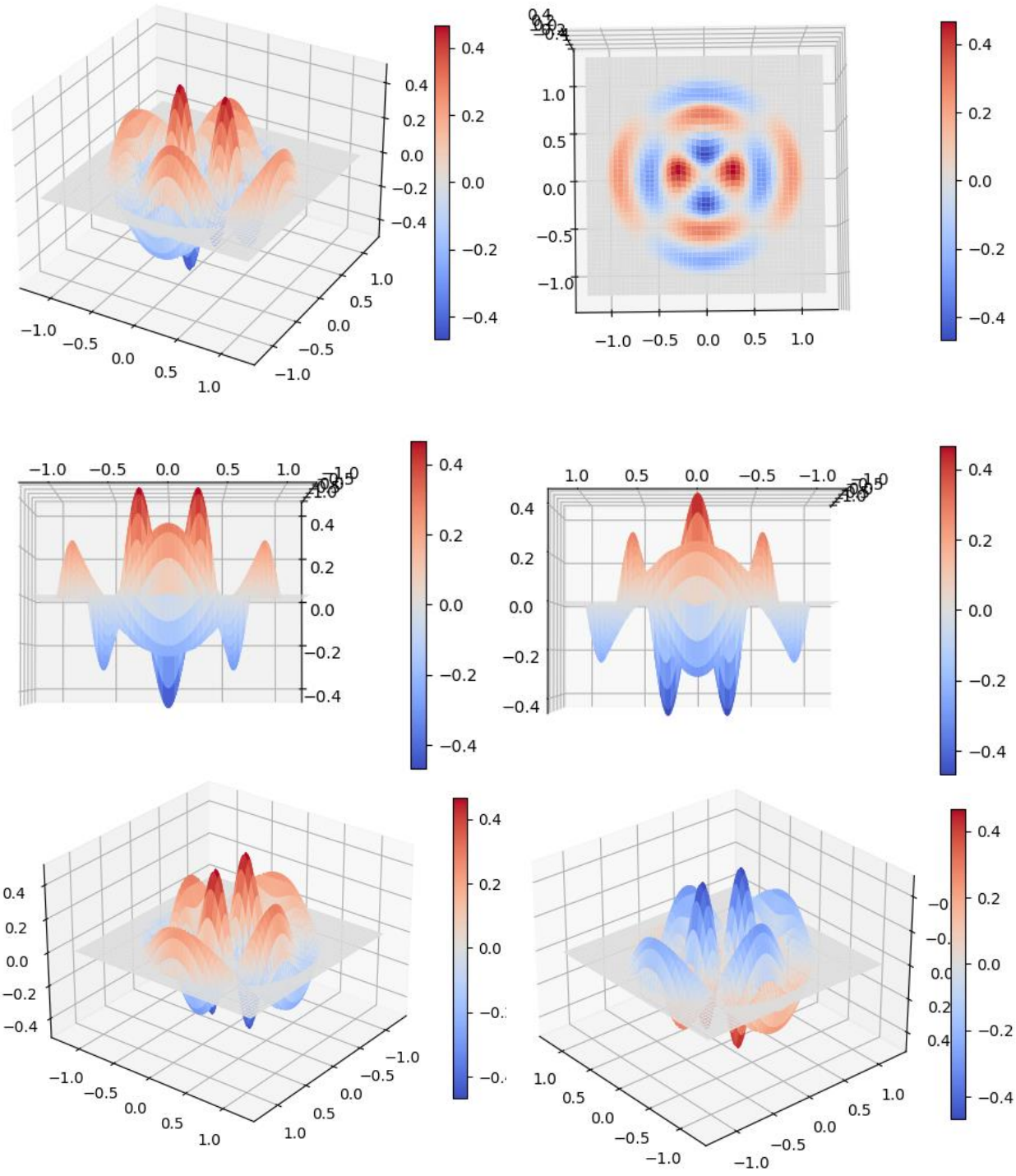


As shown in the plot above, using Simpson's reproduced an extremely close match to the Bessel functions computed by scipy. Indeed, the maximum difference between the Simpson's rule approximation and the scipy function was on the order of 10^{-15} for all 3 orders of n, and mean differences were on the order of 10^{-16}, which is the same order as rounding error due to the limitations in Python. Thus, our approximation for Bessel functions of the first kind is a very good match to those calculated by scipy.

2 (c).

As shown in the plots below, the 3 dimensional surface plot of $u_{m=3,n=2}$ satisfies the boundary condition, such that, at the "outer edge" where $r/R = 1$, we find that the surface plot is equal to 0 all around.

Figures: various angles of the surface plot of $u_{3,2}$ (equation 4 from the lab 2 instruction sheet).


0.4
0.2
0.0
−0.2
−0.4
−1.0
−0.5
0.0
0.5
1.0

Question 3 (Charlie)

A)

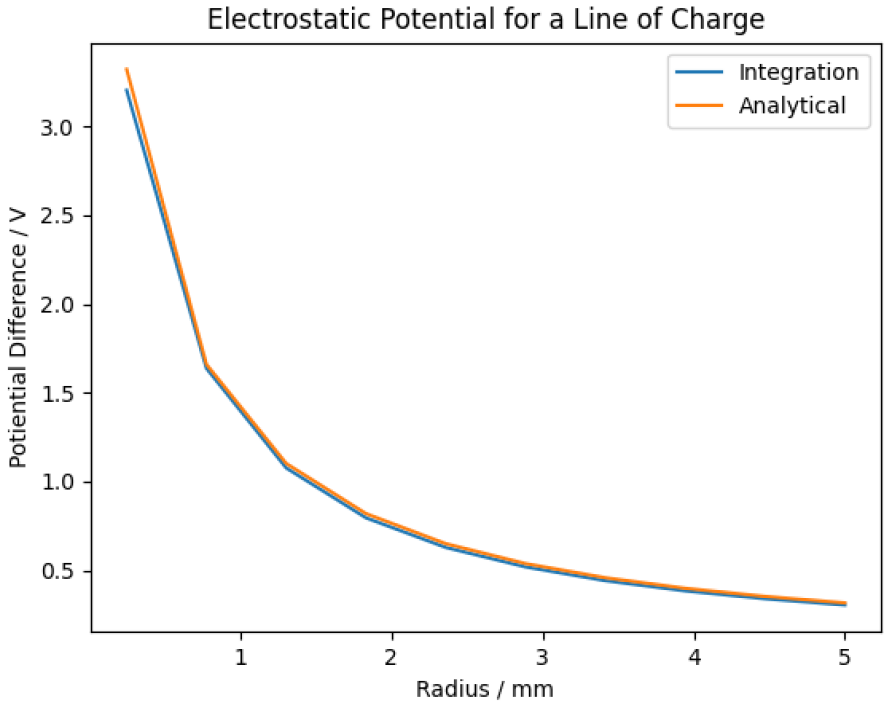


Figure: Electrostatic Potential calculated two ways using Simpson's analytical integral (8 slices) and analytical function

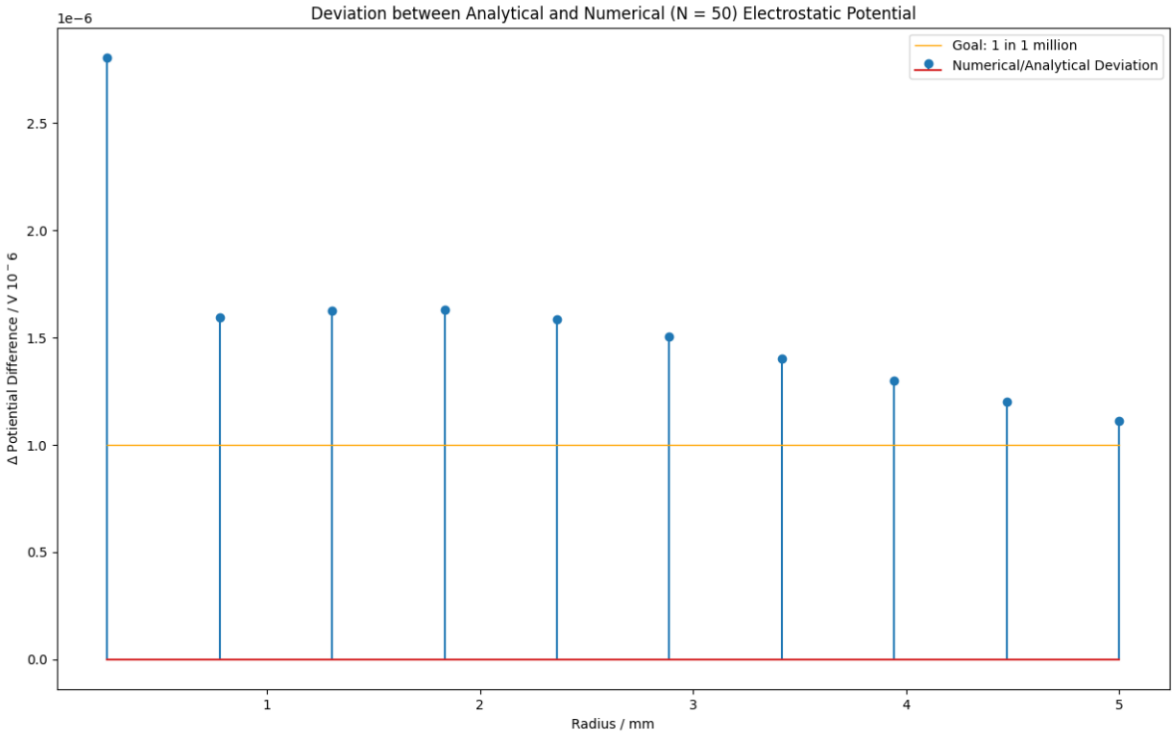


Figure: Difference between analytical and numerical function (50 slices), all approximately 1 in 1 million

Notice around N = 50 slices, the difference between the analytical calculation and the numerical integration shrinks to nearly 1 part per million. Whereas in the first figure the difference in N = 8 slices is large enough to be visible on the plot.

B)

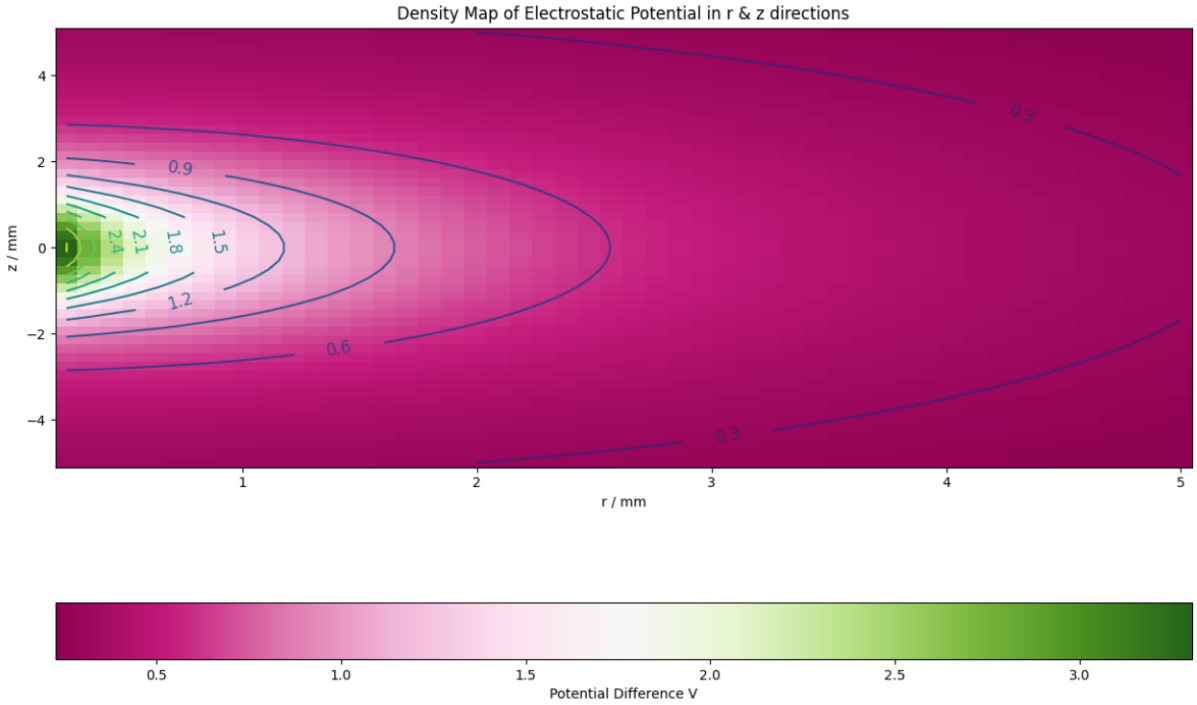


Figure: Colormap of electrostatic potential with contours across r and z axes