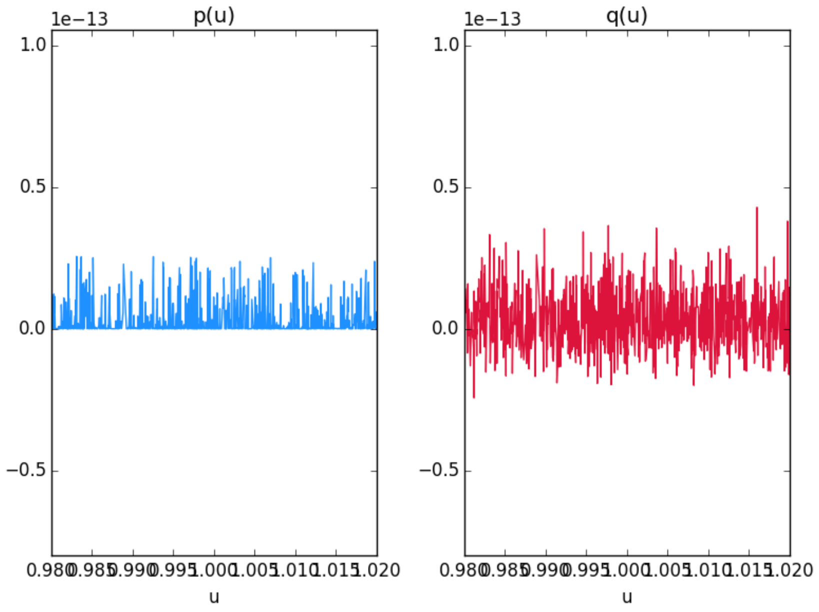
PHY407H1 Lab 2

Eric Yeung[*]
Department of Physical Sciences, University of Toronto, Toronto M1C 1A4, Canada
(Dated: September 25, 2015)

QUESTION 1

a)
See lab2_q1a.py



The plot for q(u) definitely seems noisier. This can be explained by the formulae. q(u) has a lot more operations to do [Eq. 2] if we count all the addition, multiplication, and exponents. Conversely, p(u) has significantly less operations [Eq. 1] and thus less room for error.

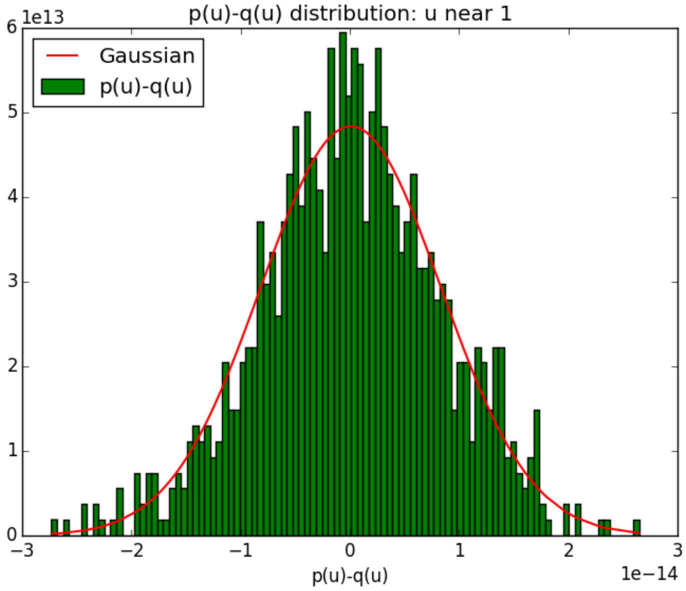
$$p(u) = (1 - u)^8 \tag{1}$$

$$q(u) = 1 - 8u + 28u^2 - 56u^3 + 70u^4 - 56u^5 + 28u^6 - 8u^7 + u^8 \tag{2}$$

I also note that the graph for p(u) rarely dips below 0 in the y-axis.

[*] eric.yeung@mail.utoronto.ca

b)
See lab2_q1b.py



The quantity $\sigma = C\sqrt{(N)}\sqrt{\bar{x^2}}$ is the standard deviation of the quantity x, or in this case p(u)-q(u). The larger σ is, the more varied the histogram would be. In this case, the value of σ should be quite large, as we have a lot of variation.

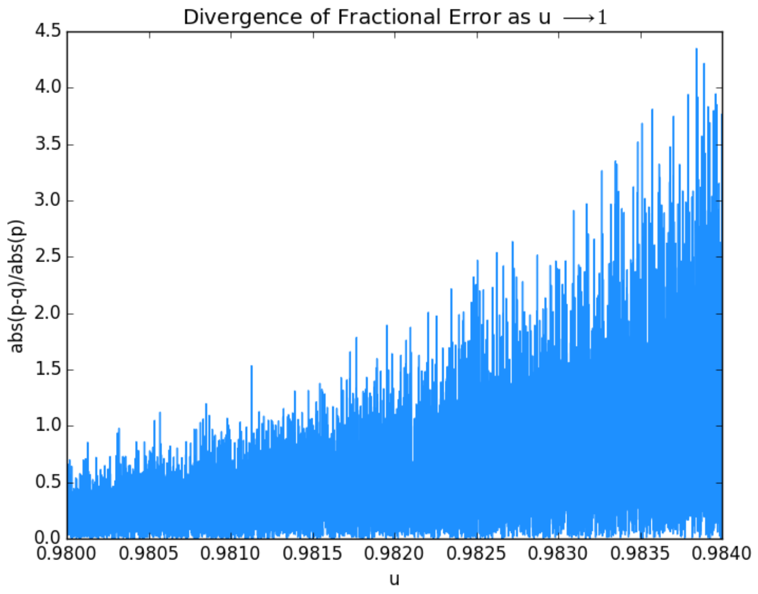
Using the std function in numpy, the standard deviation of p(u)-q(u) was found to be $\sigma = 8.1586227735e-15$. We then used $\sigma = C\sqrt{(N)}\sqrt{\bar{x^2}}$ to manually calculate the standard deviation of p(u)-q(u). Since [Eq. 1] is small in the vicinity of u=1, the error in the terms in p(u) is negligible compared to the terms in q(u). So the error in p(u)-q(u) can be reduced to simply the error in q(u).

How the terms were calculated is as follows. I took every term in q(u) that was being added and put it into an array. Then, I squared every entry of the array. Then I found the mean of these squared quantities [Eq. 3].

$$\bar{x^2} = \frac{(x_1)^2 + (x_2)^2 + ... + (x_N)^2}{N} \tag{3}$$

Then I used $\sigma = C\sqrt{(N)}\sqrt{\bar{x^2}}$ to calculate the standard deviation manually and it turned out to be $\sigma_{calc} = 3.18096220487e-15$ which is pretty consistent with numpy's result.

c)
See lab2_q1c.py

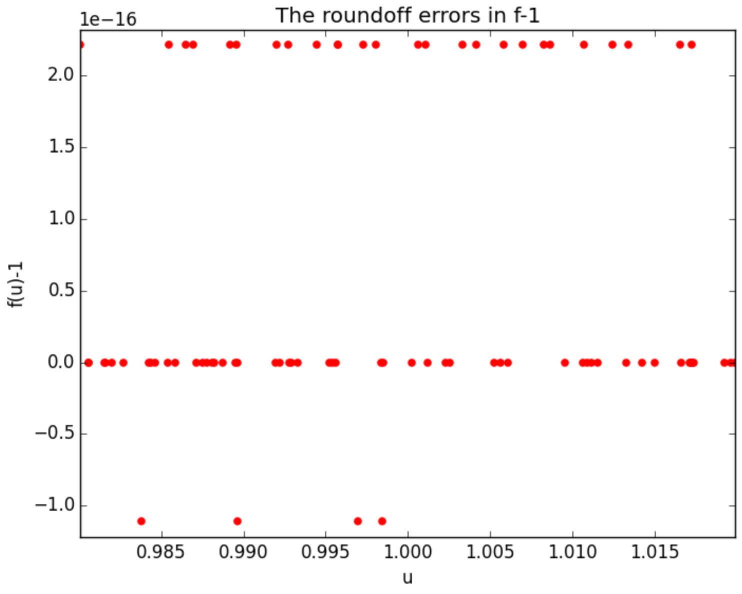


As you can see, the fractional error steadily increases as u gets closer and closer to 1.

d)
See lab2_q1d.py
The quantity f-1 was plotted vs u as a scatter plot. As one can see, there is a line of points at 0 while there is round off error, oscillating above and below the value of 0. The standard deviation was computed using numpy.std and turned out to be $\sigma = 1.29948273372e - 16$.
Then we used the error estimate in equation 4.5 in the textbook.

$$\begin{aligned}\sigma_{est} &= \sqrt{2}Cx, \text{ Replace x with f at u=1} \\ &= \sqrt{2}C\frac{1^8}{(1^4)(1^4)} \\ &= 1.41421356237e - 16\end{aligned}$$

This is pretty close to numpy's value of $\sigma = 1.29948273372e - 16$!



QUESTION 2

a)
See lab2_q2a.py

With N = 10 slices, Simpson's rule gives me 4.40042666667; the analytical value is 4.4. The fractional error is computed to be 9.69696969699e-05. With N = 100, Simpson's rule gives me 4.40000004267, with a fractional error of 9.69696969187e-09. With N = 1000, Simpson's rule gives me 4.4 (directly from python), with a fractional error of 9.69527488595e-13. The fractional error goes decreases every time N is incremented, to be expected.

With N = 10 slices, trapezoidal rule gives 4.50656, with fractional error 0.0242181818182. With N = 100 slices, trapezoidal rule gives 4.401066656, with fractional error 0.000242421818182. With N = 1000, trapezoidal rule gives me 4.40001066667, with a fractional error of 2.42424218195e-06.

It seems that Simpson's rule, in this case, is magnitudes more accurate than the trapezoidal rule. To be expected if b-a is small.

b)
See lab2_q2b.py

In this question, the integral was computed with N1 = 10 and N2 = 20 slices with the trapezoidal rule. Using equation 5.28 in the textbook error = $\frac{1}{3}(I_2 - I_1)$. The error estimate turned out to be 0.02663. The direct computation of error gives me 0.0242181818182 for the N1 = 10 slices case and 0.00605909090909 for the N2 = 20 slices case.

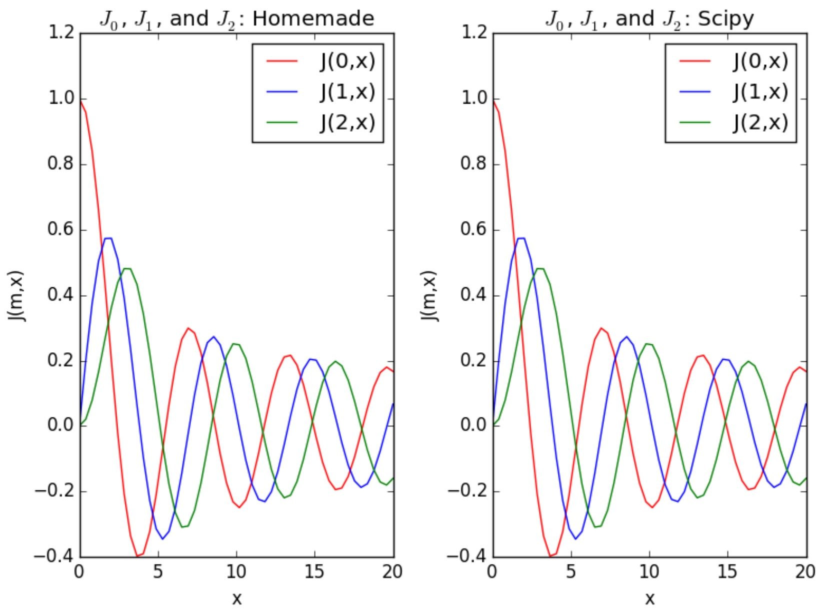
The error estimate is pretty close to the fractional error for the 10 slices case. However, it is rather far away from the 20 slices case. This is weird as the estimate correctly predicts the error for the 10 slices case, which it isn't supposed to do. It also overstates the error that it's supposed to be corresponding to. A possible reason these two values do not agree because the estimate only works if higher order can be neglected, which does not work in this case– we have a quartic function.

For example, if I increase the number of slices to 100 and 200 respectively, the error "estimate" gives 0.000266663333333 while the error for N = 200 is 6.06059090909e-05. The problem still persists.
c) See lab2_q2c.py

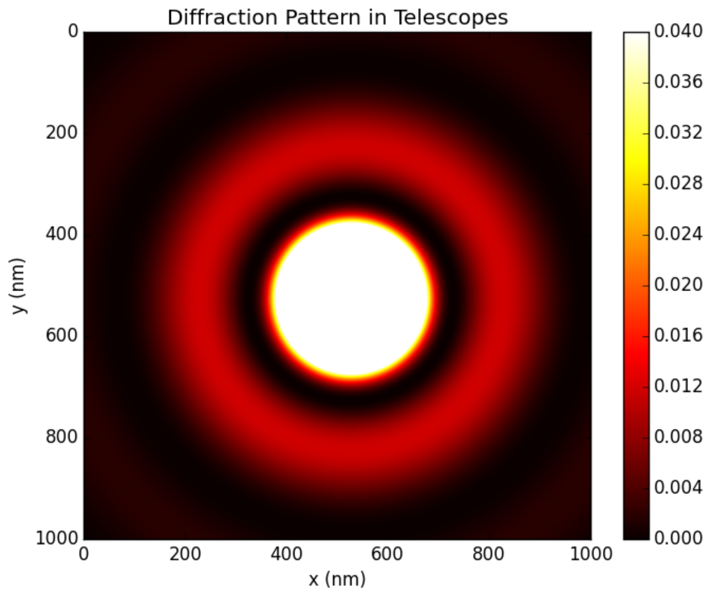
Simpson's rule with 1000 points was used to compute the integral

$$J_m(x) = \frac{1}{\pi}\int_0^{\pi} \cos(m\theta - x\sin(\theta))d\theta$$

The comparison between my manual Bessel function, and scipy's routines were plotted. I reproduced the scipy results pretty well!



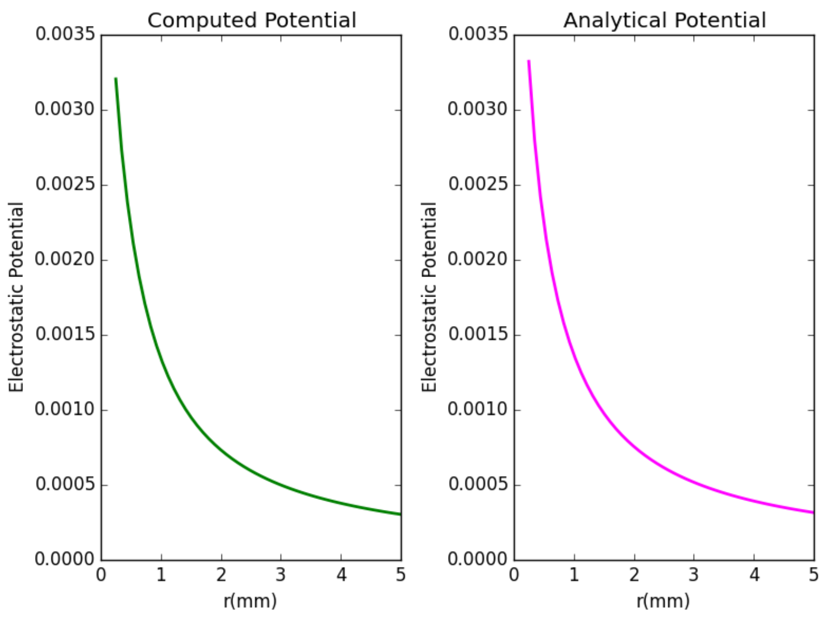
Now for the density plot.
As one can see, the intensity is at its peak and very dense at the centre. I used parameters so that the x and y span 1000 nanometres or 1 micrometer as requested in the question. I decided to keep the number of points at N = 1000 from the previous part of the question.



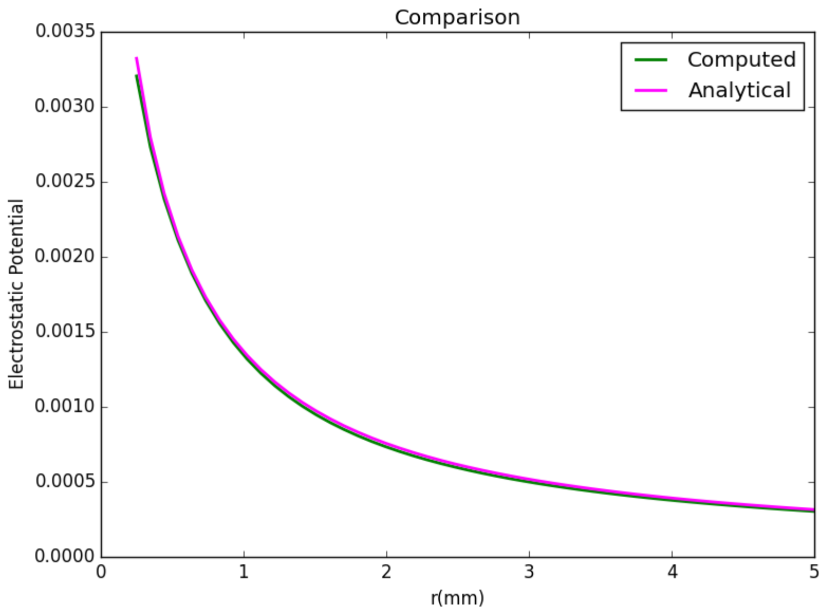
QUESTION 3

a) See lab2_q3a.py

I integrated equation 6 from the lab sheet using Simpson's rule and then compared the computational results to the analytical results. I only needed N = 8 to get a good agreement between the two results, as did Dr. Kushner. The comparison between the two are plotted below.

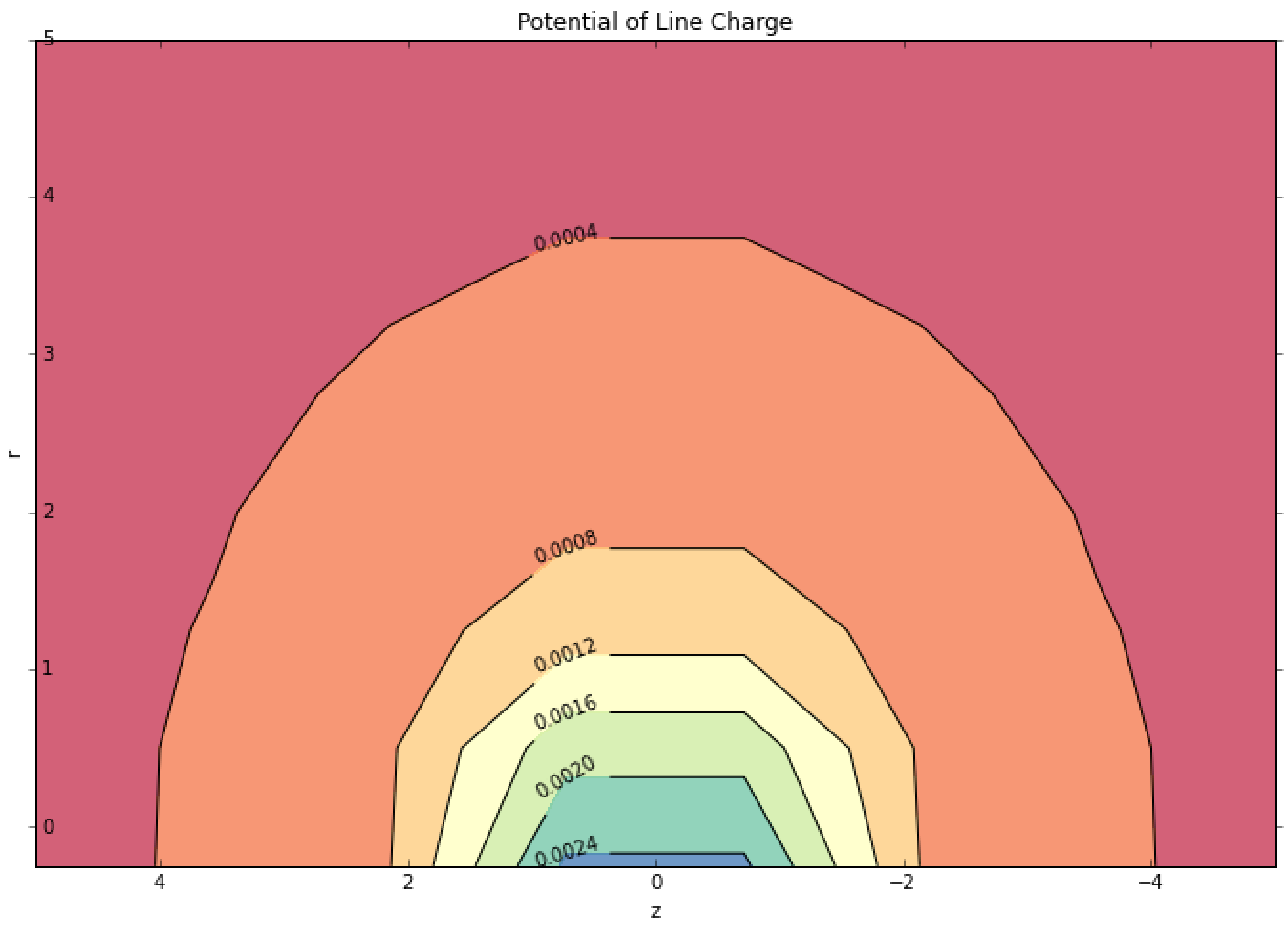


I found that, in order to have a fractional error of 1e-6, I would need around N = 50 or so.



b) See lab2_q3b.py Note to marker: This code doesn't quite run well on desktop versions of python as it returns a UnicodeWarning. However, I got this code to work in a iPython notebook. Proof will be provided if asked.

The numbers near every contour indicates the particular potential in that region.


Potential of Line Charge
0.0004
0.0008
0.0012
0.0016
0.0020
0.0024
r
z
5
4
3
2
1
0
4
2
0
−2
−4