PHY407 (Lab 2: Numerical Errors and Integration)

Utkarsh Mali[1] and Aslesha Pokhrel[1,2]

[1]Department of Physics, University of Toronto
[2]Department of Computer Science, University of Toronto

September 23, 2020

Contents

Work Allocation:
Utkarsh: Question 1, Question 3
Aslesha: Question 2

Question 1

(a) No submission required.

(b) **Here we are asked to calculate the error in each derivative by comparing the value of the numerical derivative for a specific h to the analytic derivative.** Here we print the h value, value that was obtained through numerical differentiation and the error between the numerical differentiation and the analytic value. The error taken represents the absolute value of the error.

```
    """Pseudocode for Question 1"""
# Creating h's list from 10^-16 to 10^0
# Setting initial value of x
# define forward difference
# calculate derivative at x for values of h
# computing analytic derivative
# print results as with double value
# plotting figure 1
# Answering questions
# define central difference
# calculate derivative at x for values of h using central difference
# calculating error for plotting
# plotting figure 2


OUTPUT:
[h=1e-16]    value:-1.1102230246251565 error:0.3314222415537517
[h=1e-15]    value:-0.7771561172376095 error:0.0016446658337954
[h=1e-14]    value:-0.7771561172376096 error:0.0016446658337953
[h=1e-13]    value:-0.7793765632868599 error:0.0005757802154550
[h=1e-12]    value:-0.7788214517745473 error:0.0000206687031424
[h=1e-11]    value:-0.7787992473140548 error:0.0000015357573501
[h=1e-10]    value:-0.7788014677601041 error:0.0000006846886992
[h=1e-09]    value:-0.7788008016262893 error:0.0000000185548844
[h=1e-08]     value:-0.778800790524059 error:0.0000000074526542
[h=1e-07]    value:-0.7788008216103037 error:0.0000000385388988
[h=1e-06]    value:-0.7788011724407795 error:0.0000003893693746
[h=1e-05]    value:-0.7788046770040856 error:0.0000038939326807
[h=0.0001]   value:-0.7788397166208494 error:0.0000389335494445
[h=0.001]    value:-0.7791895344301247 error:0.0003887513587199
[h=0.01]      value:-0.782629857128958 error:0.0038290740575532
[h=0.1]      value:-0.8112445700037385 error:0.0324437869323336
[h=1]        value:-0.6734015585095405 error:0.1053992245618643
```

(c) We now have to plot the error as a function of step size on a log-log plot. We found that the minimum point is indeed at $\approx 10^{-8}$. The shape of the curve is quadratic because the error is predominantly truncation errors while h is very large, and then become mostly rounding error as h approaches 10^{-16}. The "sweet-spot" between the dominating truncation error and the dominating rounding error occurs as h approaches 10^{-8}. The quadratic figure makes sense since Equation 5.91 takes the log of e^{-x^2}. This results in a $-x^2$ quadratic which is what we observed.

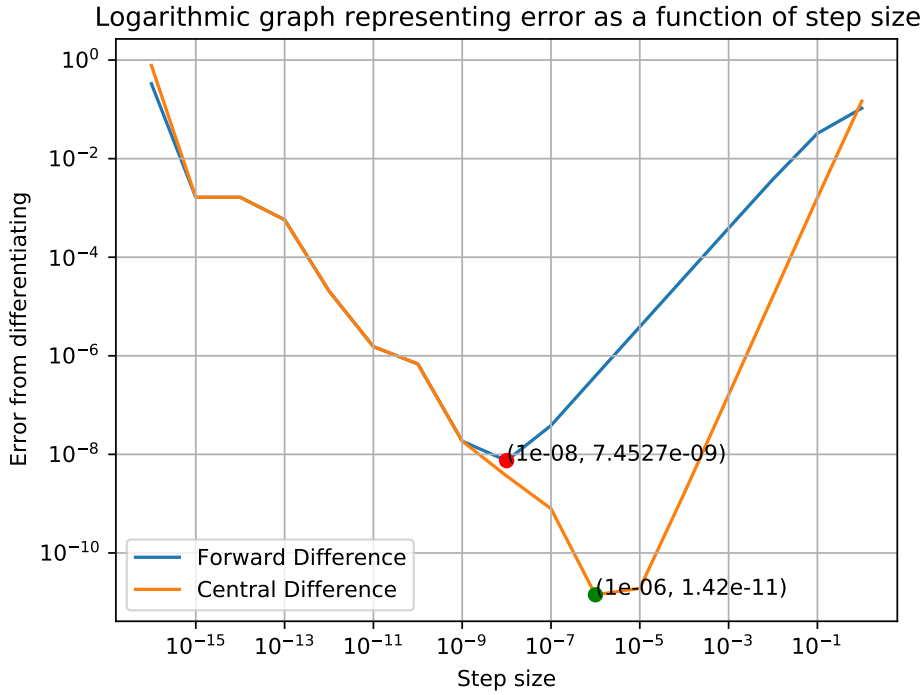

Figure 1: Here the forward and central difference method of taking derivatives can be seen with different step sizes h. When the step size is very large ($\approx 10^1$) the error from differentiating is high due to the truncation error and as the step size reduces ($\approx 10^{-16}$) the error from differentiating is also high due to the rounding error

(d) We now implement the central different method as seen in Figure 1. We observe that at small values of h, both graphs are limited by the rounding error which occurs, but as the step size crosses 10^{-8} the central difference scheme's error keeps reducing while the forward difference differentiation's error increases. The central difference is less affected by the truncation error than the forward difference error. **The central difference does not always beat the forward difference method in terms of accuracy**. At the endpoints of the graph, when h is very small and very large, the central difference method is outperformed by the forward difference method.

Question 2

(a) In this part we seek to evaluate the Dawson function given by eq. (1) using Trapezoidal and Simpson's integral rules and compare the accuracy to the `scipy's` inbuilt function `scipy.special.dawsn`.

$$D(x) = e^{-x^2} \int_0^x e^{t^2} dt \tag{1}$$

(i) Here we evaluate and compare the Dawson function for $N = 8$ slices using Trapezoidal rule, Simpson's rule and `scipy.special.dawsn`. The pseudocode and the output of the code is given below:

```
""" Pseudocode for Trapezoidal rule: """
# Get the function (f) to evaluate
# Divide the interval [a, b] into N slices
# The width of the slice is given by h = (b-a)/N
# Implement eq. (5.3) pg. 142 from the textbook
#   Accumulate the starting and ending bits in a sum as: s = 0.5*f(a) + 0.5*(b)
#   Evaluate and add the interior bits to the sum as:
#       for k from 1 to N-1:
#           add f(a+ k*h) to s
#   Multiply s by h and return


""" Pseudocode for Simpson's rule: """
# Get the function (f) to evaluate
# Divide the interval [a, b] into N slices. The number of has to be even
# The width of the slice is given by h = (b-a)/N
# Implement eq. (5.9) pg. 146 from the textbook
#   Accumulate the starting and ending bits in a sum as: s = f(a) + f(b)
#   Evaluate and add the interior bits at the odd points multiplied by 4 to the
    sum:
#       for odd k from 1 to N-1:
#           add 4 * f(a+k*h) to the sum
#   Evaluate and add the interior bits at the even points multiplied by 2 to the
    sum:
#       for even k from 1 to N-1:
#           add 2 * f(a+k*h) to the sum
#   Multiply s by (1/3)*h and return

OUTPUT:
The value of Dawson function at x=4
    using Trapezoidal integral is 0.26224782053479523
    using Simpson's integral is 0.18269096459712167
    using scipy.special.dawsn is 0.1293480012360051
```

Hence, from the output above we can see that Simpson's rule gives a much closer result to `scipy.special.dawsn` compared to the Trapezoidal rule.

(ii) Here we estimate the number of slices required to approximate the integral with an error of $O(10^{-9})$ for each method (Trapezoidal and Simpson). The Scipy's value is taken as the true value and the absolute error is used for the comparison. The in-built function `numpy.allclose` was used with `atol = 1e-09` to compare the two values where `atol` is the absolute tolerance and the function returns `True` only if the given values (true value and approximation value) differ by less than this tolerance. The output from the code is given below:

```
OUTPUT:
For the Trapezoidal rule it takes 2^12 slices to approximate the integral with
    an absolute error of O(10^(-9)).
For the Simpson's rule it takes 2^8 slices to approximate the integral with an
    absolute error of O(10^(-9)).
```

Thus, to achieve an error of $O(10^{-9})$ in the approximation of the integral it takes $N = 2^{12}$ slices for the Trapezoidal method and $N = 2^8$ slices for the Simpson's method.

The average time required for each of these methods to achieve this approximation is given below:

```
OUTPUT:
For the Trapezoidal rule it takes an average of 1.711ms to approximate an
    integral with error of O(10^(-9)).
For the Simpson's rule it takes an average of 0.111ms to approximate an integral
    with error of O(10^(-9)).
```

Hence, Simpson's rule is more accurate as well as efficient in this case.

(iii) Now, we adapt the "practical estimation errors" of the textbook (section 5.2.1, pg. 153) to both the methods to obtain an error estimation for $\mathbf{N_2 = 64}$, using $\mathbf{N_1 = 32}$. The error estimation for the Trapezoidal rule is given by **eq. (5.28)** in the textbook which is shown below:

$$\epsilon_2 = \frac{1}{3}(I_2 - I_1)$$

where I_1 is the estimate using N_1 steps and I_2 is the estimate using N_2 steps.

Similarly, the error estimation for the Simpson's rule is given by **eq. (5.29)** in the textbook which is shown below:

$$\epsilon_2 = \frac{1}{15}(I_2 - I_1)$$

where I_1 is the estimate using N_1 steps and I_2 is the estimate using N_2 steps.

The following was obtained using the equations given above:

```
OUTPUT:
The practical error estimation for the Trapezoidal rule is -0.002546568652955679
The practical error estimation for the Simpson's rule is -4.115768458675858e-05
```

Hence, the estimated error for Simpson's rule is smaller compared to the Trapezoidal rule.

(b) Here we explore the diffraction limit of the telescope.

(i) **We use Simpson's rule with $N = 1000$ slices to calculate the Bessel function given in exercise 5.4 (pg.148) of the textbook and compare the result with `scipy.special.jv` graphically.** The plots of the Bessel functions J_0, J_1 and J_2 as a function of x from $x = 0$ to $x = 20$ calculated using the Simpson's rule and in-built Scipy function is shown below:

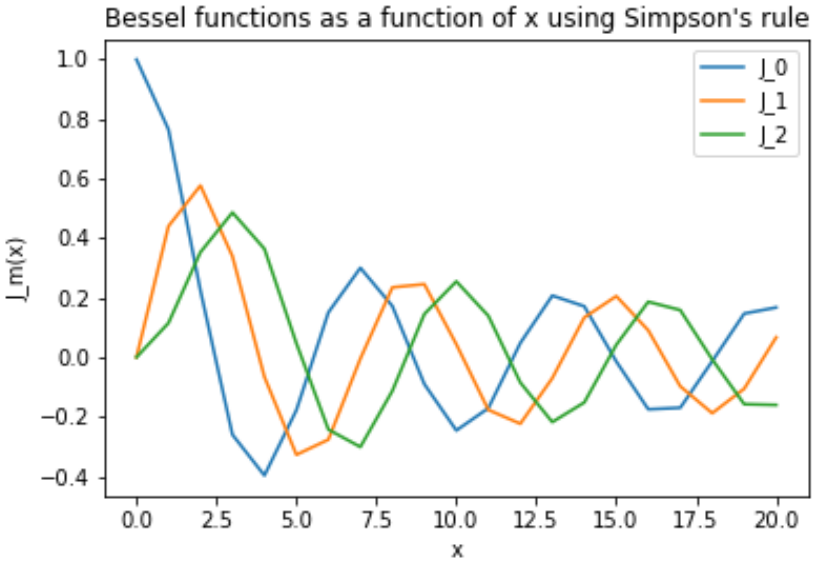

Figure 2: The plot of the Bessel functions J_0, J_1 and J_2 as a function of x from $x = 0$ to $x = 20$ calculated using the Simpson's rule.

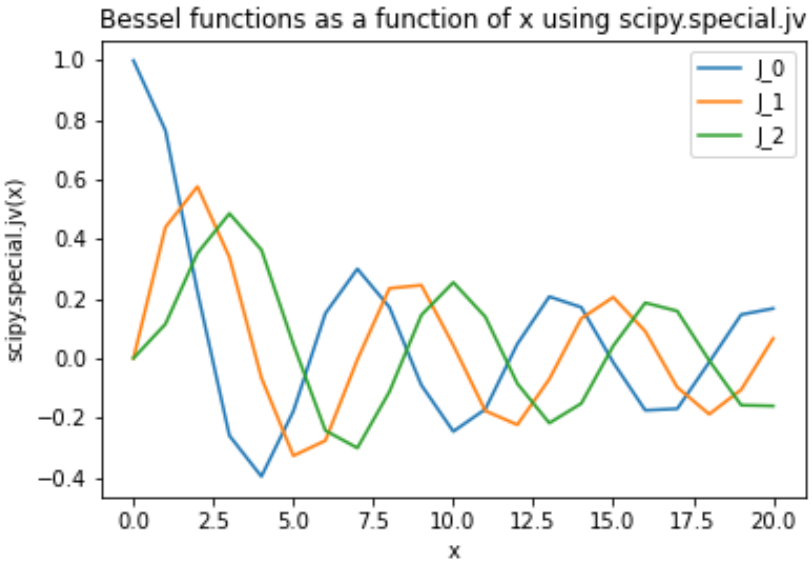

Figure 3: The plot of the Bessel functions J_0, J_1 and J_2 as a function of x from $x = 0$ to $x = 20$ calculated using `scipy.special.jv`.

Thus, from fig. 2 and fig. 3 we can see that the Bessel function I created using Simpson's rule reproduces the Scipy's routine very well.

(ii) **Here we write a program to create a density plot of the intensity of the circular diffraction pattern of a point light source with $\lambda = 500nm$ in a square region of the focal plane.** The picture covers r from 0 to $1\mu m$ and given hints are used to improve the quality of the produced image.

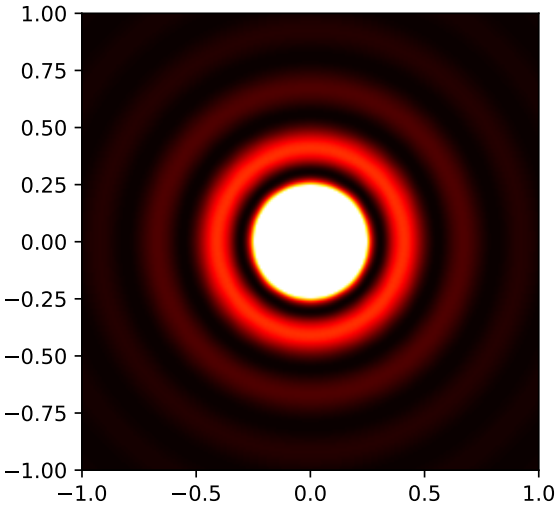

Figure 4: The plot of the circular diffraction pattern of a point light source.

Question 3

(a) No submission required

(b) No submission required

(c) **Here we use numerical integration to calculate the graph of intensity produced by the diffraction grating.** We decided to use the extended Simpsons Integration method for its accuracy and speed as highlighted in the textbook. We decided to integrate from half the slit width to the other half using the central location as our $u = 0$ location of the slit. We first compute the odd and even terms and then sum those with the computed values of I(x) as defined in Eq. 5.9 in the textbook. We then use a for loop to computer the value of this integral across x to find the value of the integral at every point so that we can plot it correctly. The for loop is run from the left side to the right side of the screen using the screen width with the center, again, centered at $x = 0$. This is then plotted as a function of position which is then ultimately used to create our heat-map. We have used `np.exp` instead of `cmath` as mentioned in the questions.

```
"""Pseudocode for Question 3a-d"""
# Import packages
# Question 1b
# slit separation
# Define transmission function q(u)
# Question 1c
# wavelength
# focal length
# total screen width
# numbers of slits
# total width
# Defining I(x)
# Simpson's method of integration
# NOTE: I will be using the extended Simson's rule
# lower bound
# upper bound
# Number of cuts
# step
# x linspace
# initialize I(x)
# for loop over x
    # initialize even sum
    # initialize odd sum
    # Odd terms
    # Even terms
    # Complete final sum for integral
# Plotting initial results
# Question 1d
# Plotting result as fancy graph including heat map
# Creating heat-map
# Label axis
# Savefig
#NOTE: Since ei,ii are similar to a-d, the pseudocode will look almost identical to
     this, and had been omitted from this attachment.
```

(d) Here we are asked to plot a density plot as a heat-map. We first plot this image as a line graph and then convert. Here we used the `plt.plot` function for the line graph and the `ax.imshow` function for the heat-map plot with `extent` used to scale the values of the heat-map to the entire range of `Ix`.

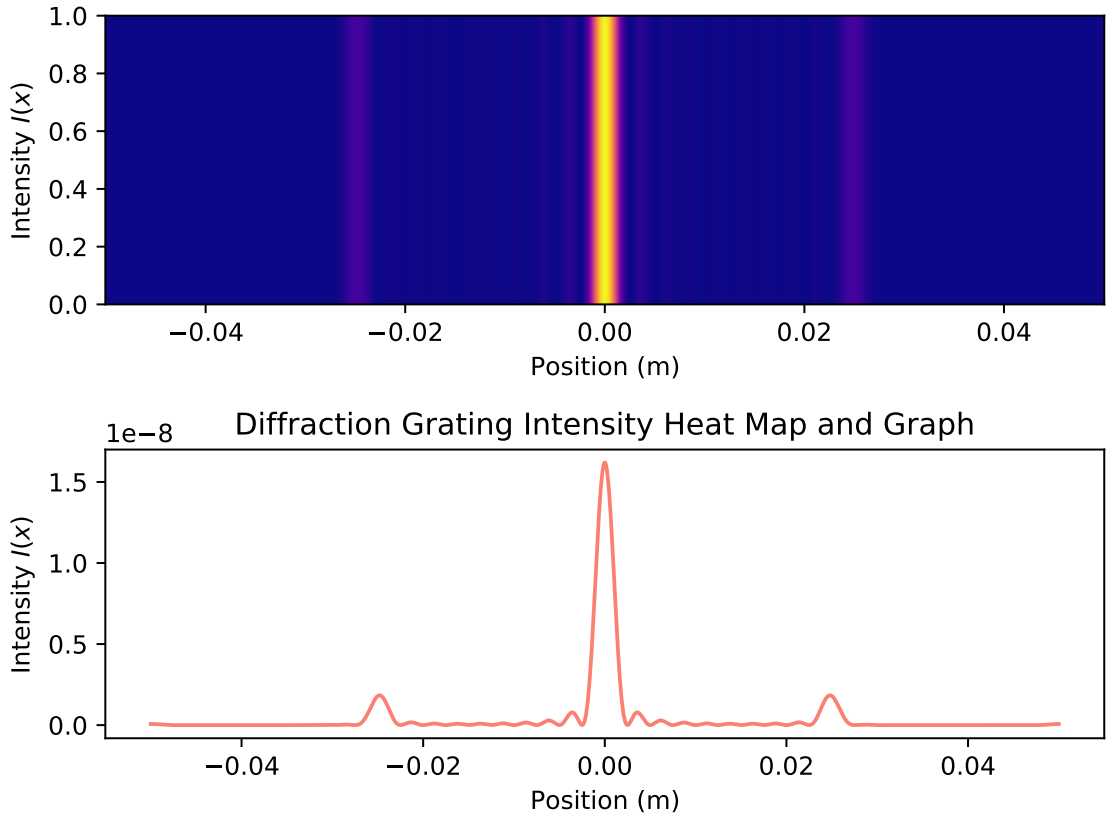

Figure 5: Here there are 2 graphs, one representing a line graph with intensity vs position and a density heat-map representing intensity vs position with red being higher intensities and blue being lower intensities.The resulting output is as expected with a characteristic diffraction grating intensity with a very sharp peak at the center and smaller peaks as the graph branches out.

(ei) Here we need to change parameters in order to accommodate for the new variable. We replace the transmission function defined from the one given in (c) to the new one with β and find that the resulting grating is similar to that of the previous question. We then run the same code as we did for Qa-d to output the plot given below.

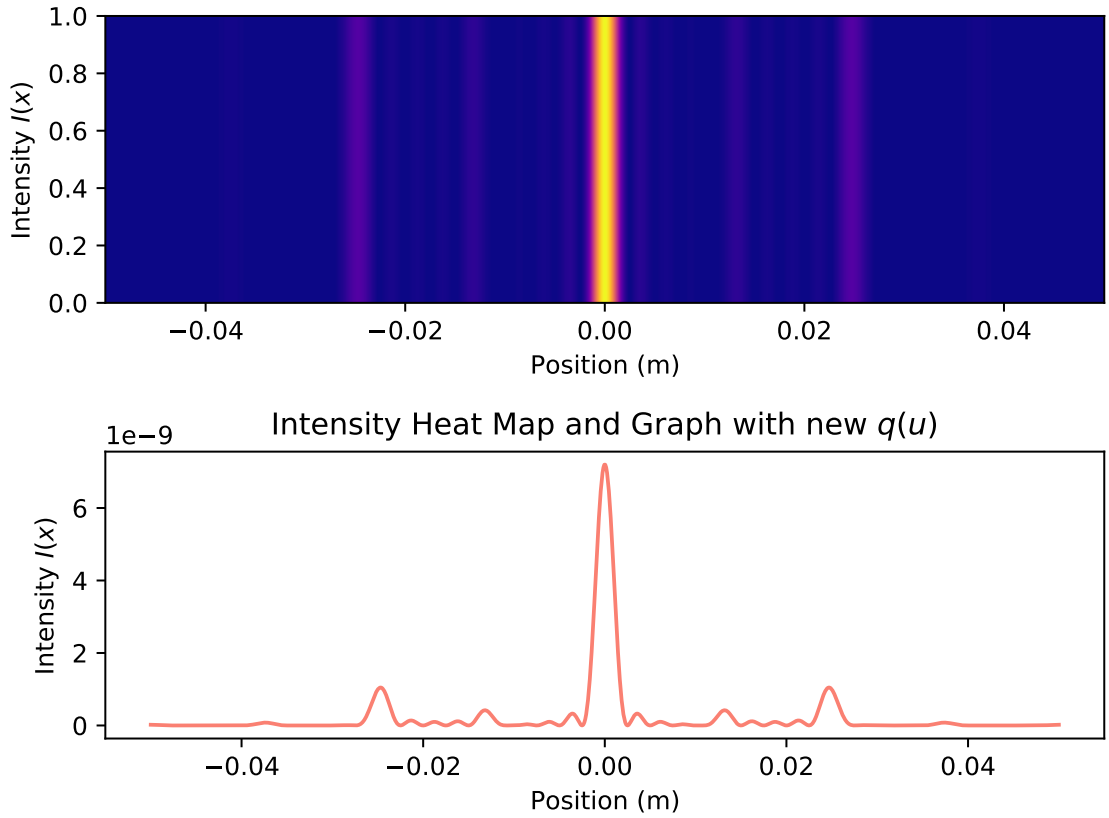

Figure 6: The 2 graphs, one representing a line graph with intensity vs position and a density heat-map representing intensity vs position. The resulting output is as expected with a characteristic diffraction grating intensity with a very sharp peak at the center and smaller peaks as the graph branches out. This is similar to Figure 5 since we are only slightly varying the transmission function, we notice a few extra peaks such as one at $x = \pm 0.015$. The value of the intensity however, has changed a lot.

(eii) Here we need to change parameters once again in order to accommodate for the new variable. We again, replace the transmission function. this time we want to simply stop all the transmission if the wave is not traveling through the slit, hence we ask the transmission function to return 100% or `1.` when the position of u is where the slit is and 0% or `0` otherwise. The resulting

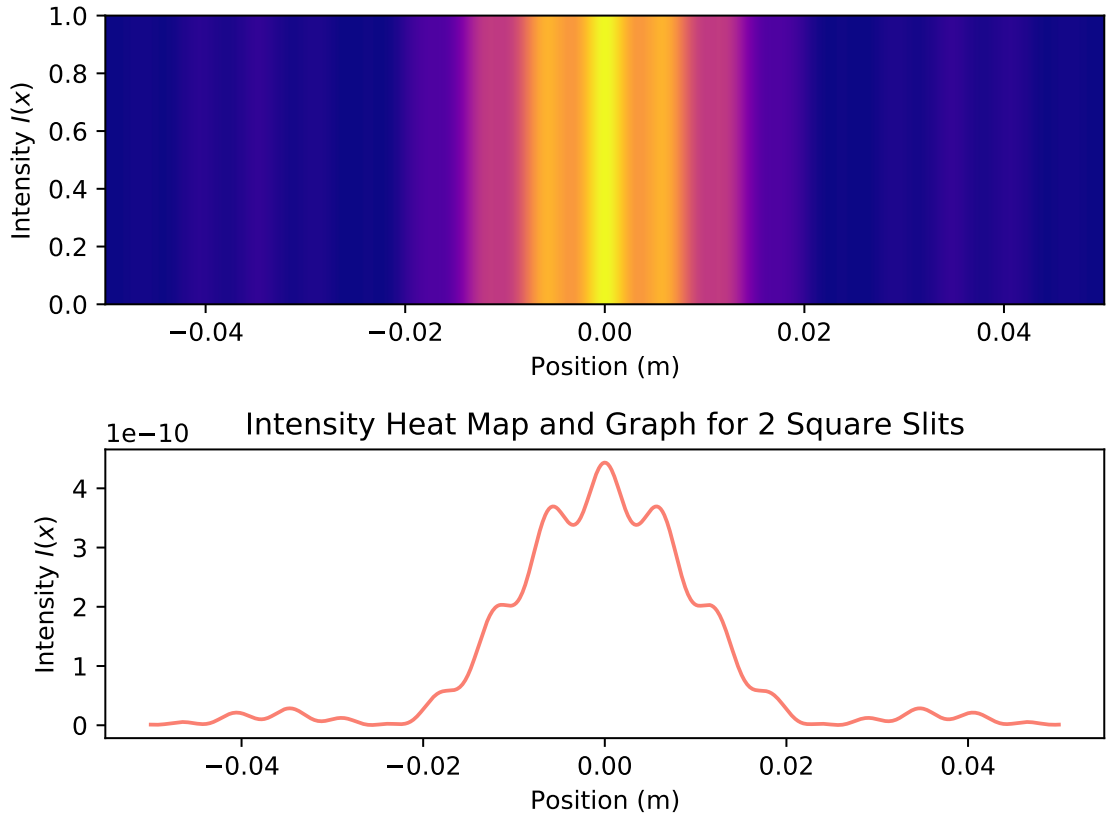

Figure 7: The 2 graphs, one representing a line graph with intensity vs position and a density heat-map representing intensity vs position. The resulting output is as expected with a characteristic diffraction grating intensity with a very sharp peak at the center and smaller peaks as the graph branches out. Unlike Figure 6 this graph is not similar to that of Figure 6, this is because as the there are only two slits which let all the waves that pass through it, we notice a pattern that is similar to single slit diffraction with a modulated double slit diffraction pattern as expected.