PHY407 Lab02

(Yonatan Eyob Q1,Q3), (Kivanc Aykac Q2)
Student Numbers: 1004253309, 1004326222

September 25, 2020

1 Numerical Differentiation Errors

1.a

Nothing to submit.

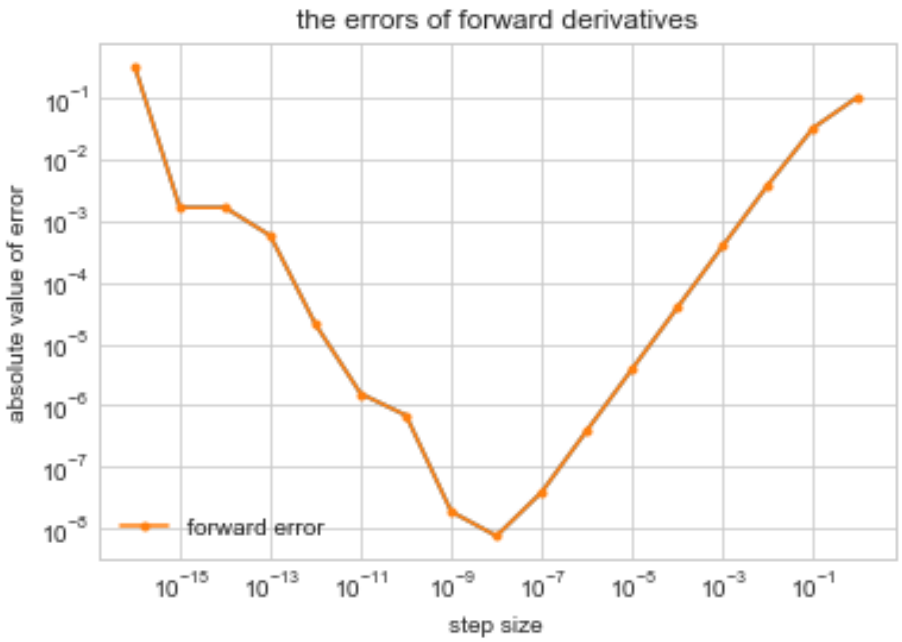
1.b

step size	forward derivative	absolute value of forward derivative error
10^{-16}	-1.11022302	3.31422242e-01
10^{-15}	-0.77715612	1.64466583e-03
10^{-14}	-0.77715612	1.64466583e-03
10^{-13}	-0.77937656	5.75780215e-04
10^{-12}	-0.77882145	2.06687031e-05
10^{-11}	-0.77879925	1.53575735e-06
10^{-10}	-0.77880147	6.84688699e-07
10^{-9}	-0.7788008	1.85548844e-08
10^{-8}	-0.77880079	7.45265416e-09
10^{-7}	-0.77880082	3.85388988e-08
10^{-6}	-0.77880117	3.89369375e-07
10^{-5}	-0.77880468	3.89393268e-06
10^{-4}	-0.77883972	3.89335494e-05
10^{-3}	-0.77918953	3.88751359e-04
10^{-2}	-0.78262986	3.82907406e-03
10^{-1}	-0.81124457	3.24437869e-02
10^{0}	-0.67340156	1.05399225e-01

1.c

Truncation error (also called approximation error) occurs in derivatives because we are only considering the first part of a Taylor series and h isn't approaching zero. For sufficiently small h, the forward derivative will not have much truncation error. Rounding error occurs when a computer attempts to subtract two numbers that are extremely close together so if h is too small the rounding error

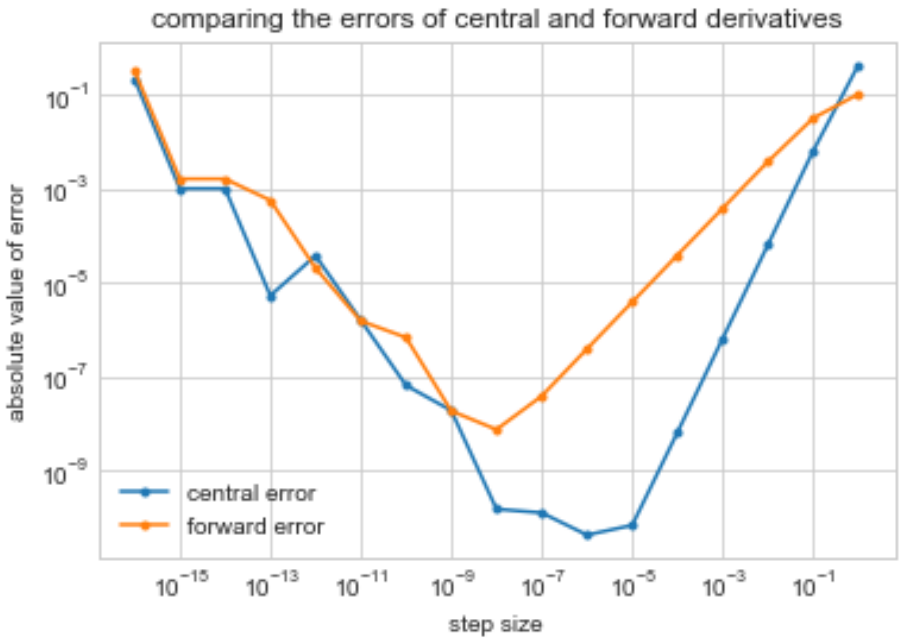
will be large. So, if h is too big the truncation error will dominate and if h is too small the rounding error will dominate.



The graph above indicates that the error of the forward derivative method becomes great for large values of h and small values of h, with h=10^{-8} being the sweet spot where the error is minimum. Considering equation (5.91) on page 190 in the textbook, it makes sense that the error gets bigger as h gets bigger because the second derivative has a greater impact on the error for larger h's. It's clear that the truncation error begins to dominate when h is greater than 10^{-8} and rounding error begins to dominate when h is smaller than 10^{-8}.

1.d

Now we will compare the error of central difference and forward derivatives for various values of h:



This graph is comparing the errors of central differences and forward derivatives dependant on the step sizes h in a range of 10^{-16} to 10^{0}, increasing by a factor of 10 each step (so there are 17 h values we are considering). From the graph we can see that central difference has less error than forward derivatives from h=10^{-1} to h=10^{-8}, h=10^{-10},h=10^{-13}, and h=10^{-15}. The errors are the same at h=10^{-9} and h=10^{-11} and forward derivative has less error at h=10^{0} and h=10^{-12}. This graph suggests that central differences are more accurate if the step size is between 10^{-1} and 10^{-8}, and the forward derivative is more accurate if the step size has a magnitude of 10^{0} or 10^{-12}. The forward and central difference seem to be equally accurate for step sizes with magnitude 10^{-9} and 10^{-11}.
Central difference has minimum error when h=10^{-6} and truncation error doesn't seem to affect the central error that much until h> 10^{-5}.

2 Trapezoidal and Simpson's Rules for Integration

2.a

Dawson function:

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

2.a.i

In this part, goal is to compute the equation (1) for $N = 8$ slices in three different methods and compare the methods: Trapezoidal, Simpson's Rule and the `scipy.special.dawsn`.
Pseudo code for this part as follows:

- Define the Dawson function
- Define the Trapezoidal method function for integration for this case
- Define the Simpson's Rule method function for integration for this case
- Define the function to get the `scipy.special.dawsn` value
- Set N for the number of slices (has to be even because Simpson's Rule requires so)
- Print the computed values

After doing the coding along the lines of the above pseudo-code, result (5 digits after decimal point) is as follows:

```
For N=8 slices and x=4,
Trapezoidal Rule gives: 0.26225
Simpson's Rule gives: 0.18269
The scipy.special.dawsn function gives: 0.12935
```

As can be seen above, for $x = 4$ and $N = 8$ slices, three of the methods give distant results. With the reading from the textbook in mind, this is suspected to be because of the number of slices used. With more slices, the methods should converge/agree at a range of values. To further investigate this suspicion, below exercises work on this.

2.a.ii

In this part, goal is to To achieve this, a "while-loop" method is interpreted. The pseudo code for this part is as follow (building on top of the previous section 2.a.i):

- Define the relative error function
- Define a function to find the number of slices (N) for a given n
- Set a while loop that will go over different values of n with iteration $n+1$ until an error of $O(10^{-9})$ from the true value is reached (treating the `scipy.special.dawsn` as the true value)
- Set the initial n value
- Operate the above function with two different methods' functions (Trapezoidal and Simpson's Rule)

- At the same time, store the error values and their corresponding n values to plot
- Print the result
- Plot relative error vs n
- With the determined n value that gives the desired error, time how long it takes program to do the integral
- Print the time results for the two methods

After doing the coding along the lines of this pseudo code, all of the printed outputs for 2.a.ii are as follows:

```
For N=8 slices and x=4,
Trapezoidal Rule gives: 0.262248
Simpson's Rule gives: 0.182691
The scipy.special.dawsn function gives: 0.129348

With the Trapezoidal method for integration on the Dawson function,
the n value (N=n^2) that gives the desired error is 19

With the Simpson's Rule method for integration on the Dawson
function, the n value (N=n^2) that gives the desired error is 11

After 25 repetitions the mean times it took for the methods to
compute the Dawson function with their appropriate number of slices
that gives the desired error (10^-9):
Trapezoidal method --> 0.0000706959 sec
Simpson's Rule method --> 0.0000498390 sec
```

As can be seen, the first set of findings show that for only $N = 8$ slices, the difference between the methods are fairly large. Therefore as the instructions suggest a need for a standard in error is exists here.
To do so, there has to be a "true" value. For this, `scipy.special.dawsn` function's output for $x = 4$ is going to be used (`scipy.special.dawsn(4)` = 0.129348). Taking $N = 2^n$, and iterating the computations each time with $n_{new} = n_{old} + 1$ (starting from $n = 3$) until a relative error of $O(10^{-9})$ is reached gives the results $n_{Trapezoid} = 19$ and $n_{Simpson} = 11$. For better clarity, the figure 1 shows how the magnitude of the relative error decreases as n get higher. The relation is logarithmic, therefore a logarithmic scale for the y-axis causes a linear trend on this plot.
Similarly, for the Simpson's Rule method as well, the magnitude value of the relative error has been plotted against the n values on figure 2. Similarly, a logarithmic scale on the y-axis causes a linear trend on this plot.

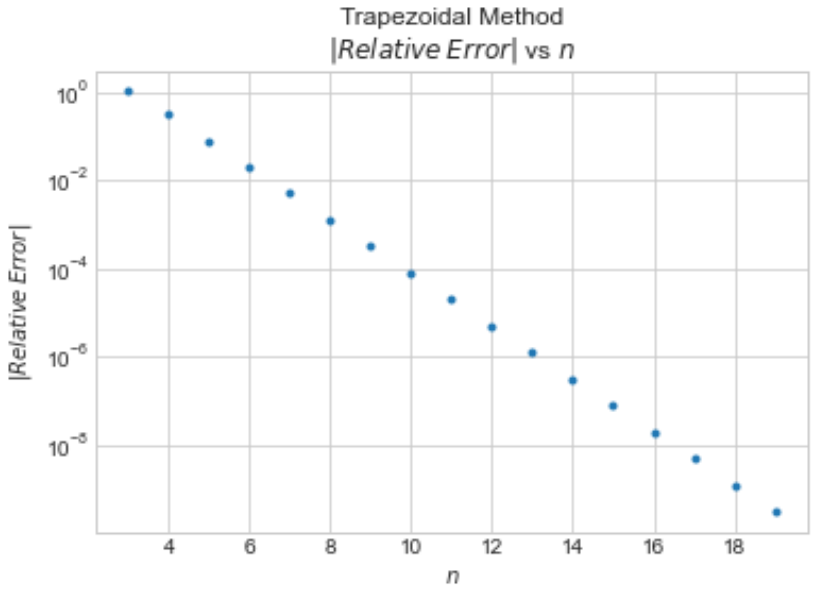


Figure 1: Magnitude of the relative error values (`scipy.special.dawsn` treated as the true value) vs n values on logarithmic y-axis for the Trapezoidal method

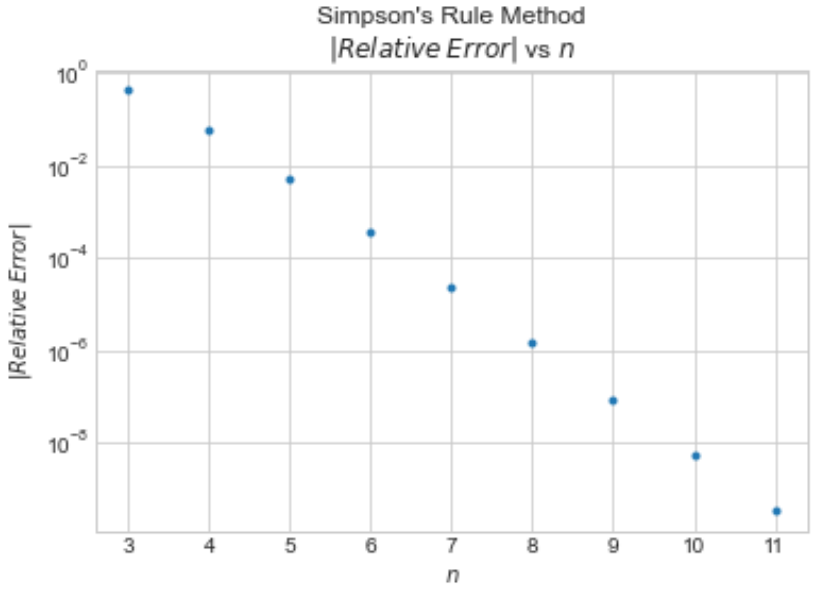


Figure 2: Magnitude of the relative error values (`scipy.special.dawsn` treated as the true value) vs n values on logarithmic y-axis for the Simpson's Rule method

Also, the time taken to calculate the integrals with the found n values for each of the methods is calculated. While the Trapezoidal method takes $\sim 71\mu sec$, the Simpson's Rule method takes $\sim 50\mu sec$. So, the Trapezoid method is slower than the Simpson's Rule method by a factor of 1.42.

2.a.iii

In this part, goal is to quickly calculate the error on both methods with the practical method using the equations 2.

$$\begin{aligned} \epsilon_{Trapezoid} &= \tfrac{1}{3}(I_2 - I_1) \\ \epsilon_{Simpson} &= \tfrac{1}{15}(I_2 - I_1) \end{aligned} \tag{2}$$

where I_i represents the computed value of the integration with N_i slices; and I_{i+1} corresponds to $N_{i+1} = 2 * N_i$. In this exercise N_1 value is taken to be 32, which implies that $N_2 = 64$ slices. Findings are as follows:

```
Using the practical estimation method, the error for two methods:
Trapezoidal method --> -0.0025465687
Simpson's Rule method --> -0.0000411577
```

As can be seen, with the same number of slices, Simpson's Rule method converges to a small ϵ value faster than the Trapezoidal method.

2.b

Bessel functions, $J_m(x)$ are going to be investigated on this section with an example (The diffraction limit of a telescope).

2.b.i

Goal is to plot Bessel functions, $J_m(x)$, with $m = 0, 1, 2$ and x from $x = 0$ to $x = 20$ with 1000 points. Pseudo code is as follows:

- Define the Bessel function that accepts m and x values, define the integrand inside this function and utilize the Simpson's Rule for integration here
- Plot all three $J_m(x)$ with $m = 0, 1, 2$

After doing the coding, figure 3 is plotted.

Not much can be commented from here other than the different shapes of the function depending on m.

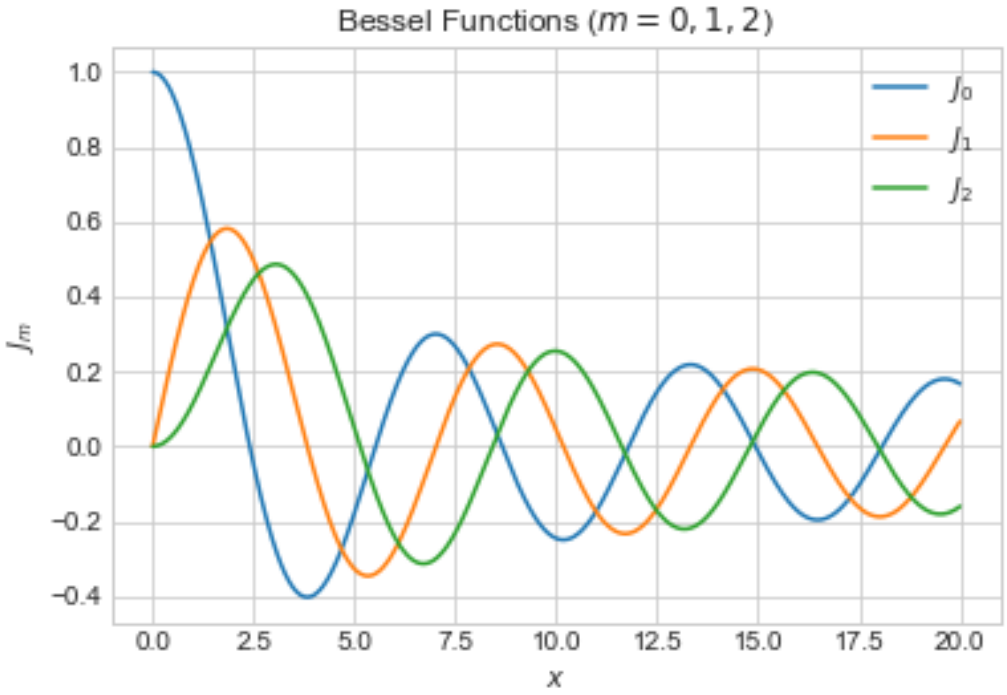


Figure 3: Bessel functions, $J_m(x)$, with $m = 0, 1, 2$ and x from $x = 0$ to $x = 20$ with 1000 points

2.b.ii

Goal is to compare the above findings to `scipy.special.jv` outputs with the same m and x values.
The Bessel function output of the `scipy.special.jv` is plotted on Figure 4.

By looking at Figure 3 and Figure 4, it is hard to differentiate any difference between the two methods. A further investigation is needed at this point.
To achieve this goal, the graph of the difference between the `scipy.special.jv` and the manually calculated (part 2.b.i) values against x is plotted. Figure 5 shows this.

By looking at Figure 5, user can be satisfied since the "$1e-15$" indication at the top of the y-axis indicates that the function that was defined is only off from the `scipy.special.jv` value by $O(10^{-15})$.
To be specific the root-mean-square values for each differences is calculated. The results are as follows:

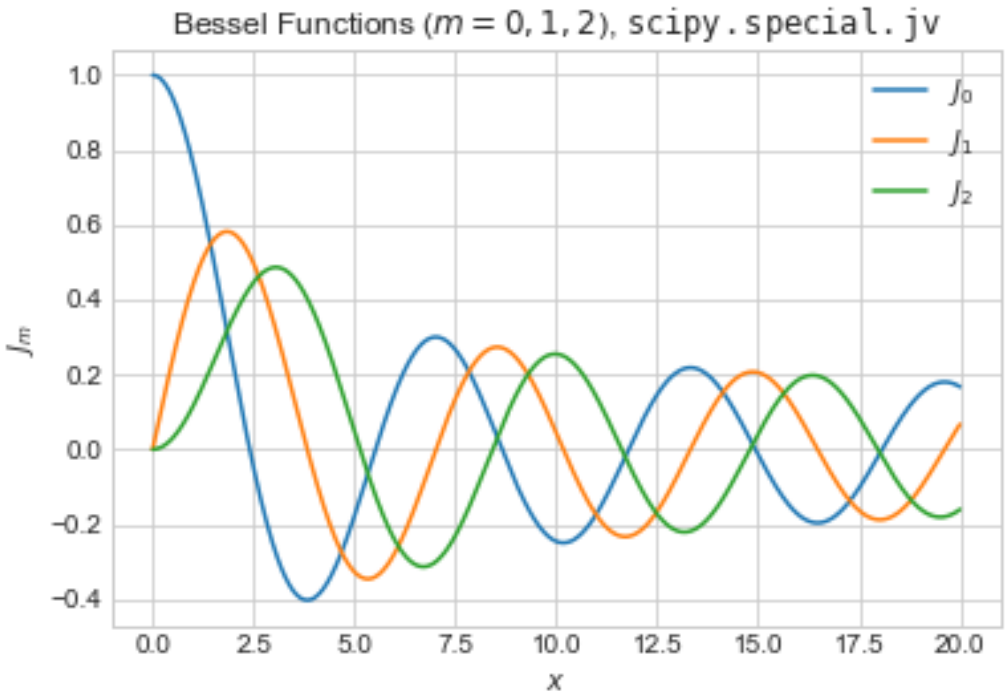


Figure 4: `scipy.special.jv(m, x)` with $m = 0, 1, 2$ and x from $x = 0$ to $x = 20$ with 1000 points

```
The RMS value for the differences between two Bessel function
methods:
J_0 --> 4.466739434314852e-16
J_1 --> 3.3366196263598283e-16
J_2 --> 3.14163500906148e-16
```

Therefore, the overall offset is close to the error constant, $C = O(10^{-16})$. This shows that the manually written function is at an high accuracy if we treat the `scipy.special.jv` as the true value.

2.b.iii

In this part, goal is to plot the intensity of the circular diffraction pattern over a grid that has sizes of $2\mu m$ from both sizes. The formulae is as follows:

$$\begin{aligned} I(r) &= (\tfrac{J_1(kr)}{kr})^2 \\ J_m(x) &= \tfrac{1}{m}\int_0^\pi \cos(m\theta - x\sin(\theta))d\theta \end{aligned} \tag{3}$$

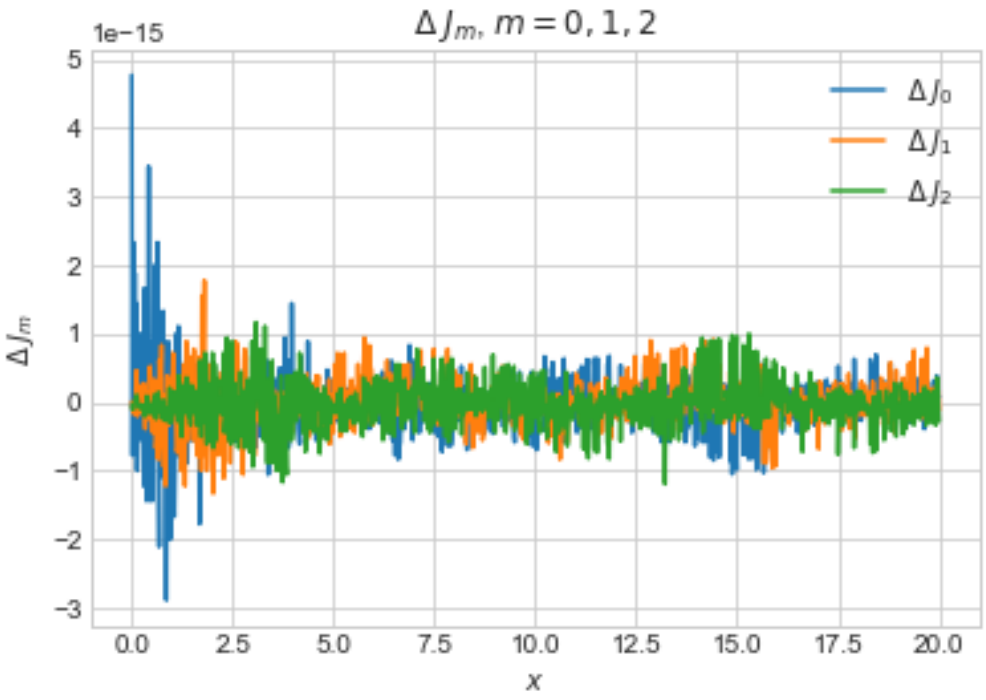


Figure 5: Difference between Bessel functions calculation methods.

where $k = 2\pi/\lambda$ and r is the distance from the centre.
The pseudo code is as follows:

- Set the variables
- Define the intensity function 3
- Define a function to calculate radius with given x and y
- Create an empty matrix that will store the information of intensities
- Create a nested loop that will iterate the above calculations for each x and y, and will store the information of intensities
- Plot the intensities with their corresponding x and y values

After doing the coding, figure 6 is plotted.

As can be seen from Figure 6, an expected circular pattern of intensities is present. Circular fringes are a result of the Bessel function in equation 3. The intensity decreases as r (radius) gets larger.

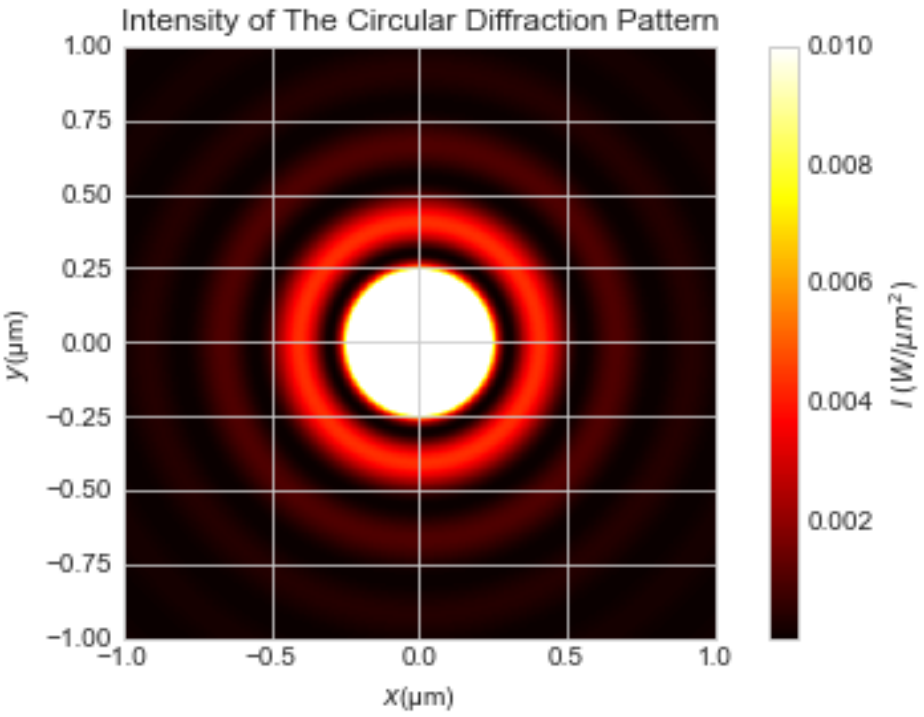


Figure 6: Intensity values for each x and y values using equation 3. $150x$ and $150y$ values were used.

3 Light Diffraction Patterns

3.a

Nothing to submit.

3.b

Nothing to submit.

3.c

- assign variables to the given values
- set up an integrand function dependant on u and x, u is the value we wish to integrate over and x is the position on the screen (x=0 is the center of the screen). incorporate the transmission function from question (3b), which is dependant on u, in this integrand

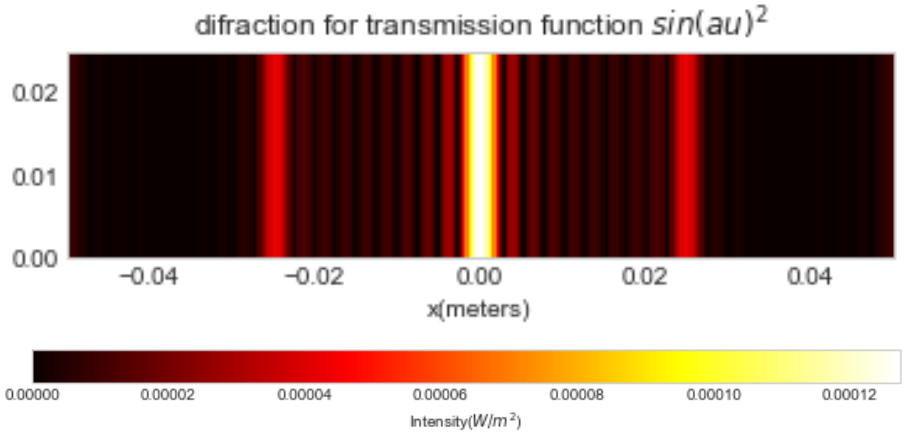
– integrate the integrand function using the Simpson rule

– create an array with entries being the Simpson integral of the integrand for different x values (this will be useful for question (3d)

In the density plot on page 207 of the textbook, the light looks most intense at x=0 (middle of the screen), then the intensity acts like a damped oscillator as x gets more positive or negative, getting dark then bright (less bright than the previous peak) and so on. The intensity pattern is symmetrical as well. So, similarly to how we can decipher the pattern of a sin function with as little as 20 points, we can also figure out the intensity pattern on the screen with few x values. When picking x values, we recognized that the screen is 0.1 meters wide so the max and min x values will be -0.05m and 0.05m. Therefore our choices of x should be between -0.05m and 0.05m in equal increments for the pattern to reveal itself. We chose 250 increments because the graph doesn't become more clear with more x values.

3.d

From here on out we will be using the intensity function I(x)=$\|\int_{-w/2}^{w/2}\sqrt{q(u)}e^{i2\pi xu/\lambda f}du\|^2$ to create density plots to create a visual of the expected diffraction patterns that should form from the given setups. All of these density plots below are the non-squared version of the intensity function I(x)=$\|\int_{-w/2}^{w/2}\sqrt{q(u)}e^{i2\pi xu/\lambda f}du\|$ (which we were told we could do on the handout). This first density plot has transmission function q(u)=$sin^2\alpha u$, with 10 slits that are seperated by $20\mu m$. The focal length is 1m, the width of the screen is 10cm, and the wavelength of the light is 500nm.



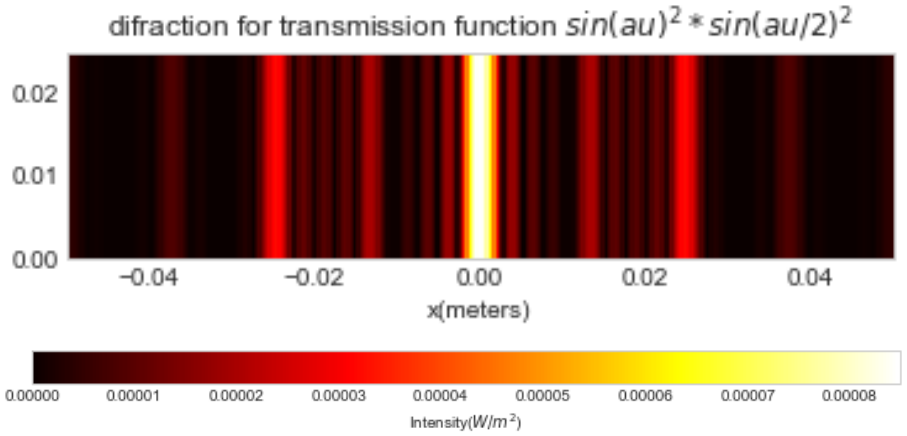
At x=0 the intensity is at it's peak, and as $\|\ x\ \|$ increases, the intensity sinusoidally fades like a damped oscillator then there is an intensity spike at about x=-0.025 and x=0.025 (just like the image on page 207 of the textbook!). The

max intensity on this density graph is $0.0001273239655031534 3W/m^2$.

3.e

3.e.i

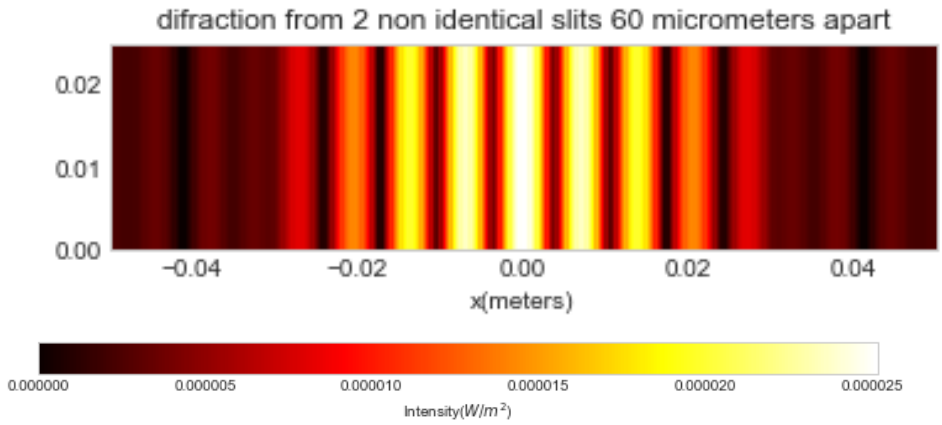
In this next density graph, the transmission function was changed to q(u)=$sin^2\alpha u sin^2\beta u$ with β=$\alpha/2$.



The diffraction pattern looks similar to the previous density plot but there seems to be 4 intensity spikes at about x=-0.3,-0.1,0.1,0.3. The max intensity has lowered as well. The max intensity on this density graph is $8.488264597196078e-05W/m^2$.

3.e.ii

In the final density plot below, we change the grating to have only two non-identical slits. The intensity of light that goes through one of the two slits is 100 percent but the intensity of light that goes through any other part of the grating is zero percent. One slit is $10\mu m$ wide the other is $20\mu m$ wide and they are $60\mu m$ apart from each other (so the grating is at least 90 micro meters wide).



The diffraction pattern looks much neater in this density graph but the peak intensity is much lower than the previous two density plots at their peak. The max intensity in this final density graph is 2.5139999999999996e-05W/m^2.