Lab 2 Report

Q2.

a)

i.

Using the trapezoid rule, I obtained around 0.2622 for the Dawson integral computed at x = 4 with N=8 slices.

Using Simpson’s rule, I obtained 0.1827 for the Dawson integral computed at x = 4 with N=8 slices.

The actual (or at least extremely accurate) value of the Dawson integral at x = 4, given by scipy.special.dawsn(4), is 0.1293.

Comparing these numbers, we see that integrating with Simpson’s rule gave a closer prediction to the actual value than integrating with the trapezoid rule. This agrees with the fact that Simpson’s rule is generally more accurate than the trapezoid rule for equivalently determined slices.

ii. I varied N1 and N2 in the code, which are the number of slices in the trapezoid rule and Simpson’s rule integration, respectively, until I got an error in the range of 10-9. I used the equation: calculated value – actual value

For N1 = 100000, I got an absolute error of 1.0667 x 10-9 with the trapezoid rule.

For N2 = 940, I got an absolute error of 1.0199 x 10-9 with Simpson’s rule.

This shows that Simpson’s rule requires significantly less slices than the trapezoid rule to achieve a certain accuracy. For timing each method, I recorded the time elapsed for each method 1000 times, then took the average of these 1000 time recordings.

For the trapezoid rule and N1 = 100000 (to get approximately 10-9 error), the calculation takes around 0.2095 s.

For Simpson’s rule and N2 = 940 (to get approximately 10-9 error), the calculation takes around 0.002167 s.

For the built-in function scipy.special.dawsn(), the calculation takes around 5.9819 x 10-6 s.

Simpson’s rule was approximately 100 times faster than the trapezoid method. This makes sense because there were around 100 times less slices required to get the desired error, and the time each method takes is proportional to the number of slices N. The built-in function was much faster than Simpson’s rule.

iii. Using the methods described in the “Practical Estimation of Errors” section of the textbook (p. 153), and starting with N1 = 32, I got:

0.0025465686529556795 as the absolute error for the trapezoid rule, and

4.115768458675858 x 10-5 as the absolute error for Simpson’s rule.

b)

5.4 (a)

After doing exercise 5.4 (a), I compared my Bessel function (obtained with Simpson’s rule for N = 1000 slices) and those in scipy.special.jv. My Bessel functions were extremely accurate, and the error was tiny! I first plotted each of the two J0(x), J1(x), J2(x) functions (mine and scipy.special.jv(i, x) where i=0,1,2) against each other, shown in Figures 1,2,3. In each plot, the two curves look very similar to each other and basically completely overlap. To understand the differences between the curves, I plotted the difference between my Bessel functions and those defined by scipy.special.jv, shown in Figures 4,5,6. These errors are on the order of 10-15, which is very impressive, considering only N = 1000 slices were used.

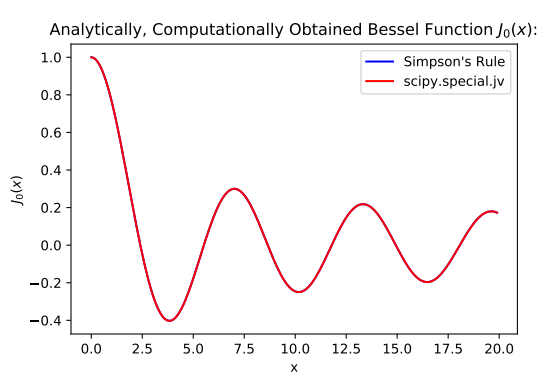


Figure 1: Plotting the analytically (scipy.special.jv) and computationally (Simpson's rule) determined Bessel function J0(x). There is barely any discernible difference in this graphical comparison.

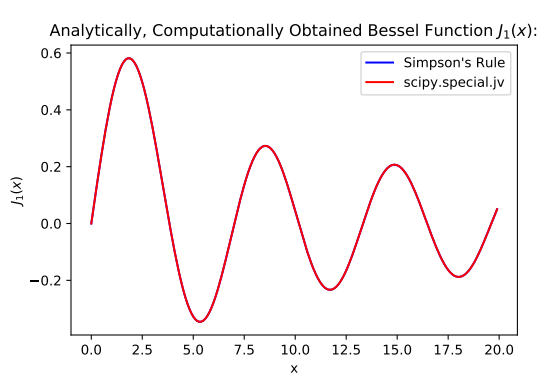


Figure 2: Plotting the analytically (scipy.special.jv) and computationally (Simpson's rule) determined Bessel function J1(x). There is barely any discernible difference in this graphical comparison.

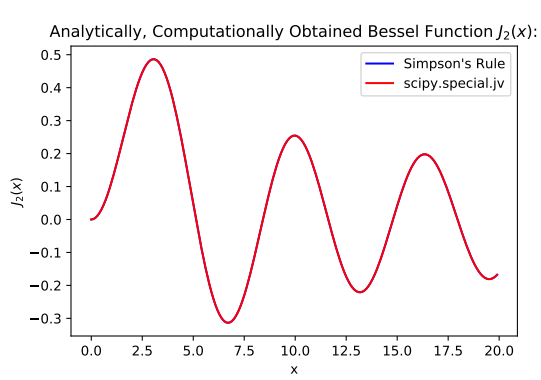


Figure 3: Plotting the analytically (scipy.special.jv) and computationally (Simpson's rule) determined Bessel function J2(x). There is barely any discernible difference in this graphical comparison.

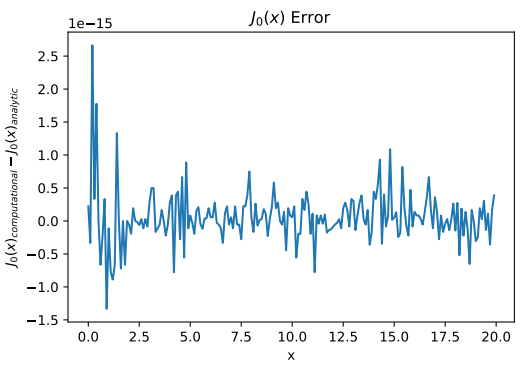


Figure 4: Plotting the error between the computational (Simpson's rule) and analytic (scipy.special.jv) Bessel function J0(x). The error is on the order of 10-15, quite impressive.

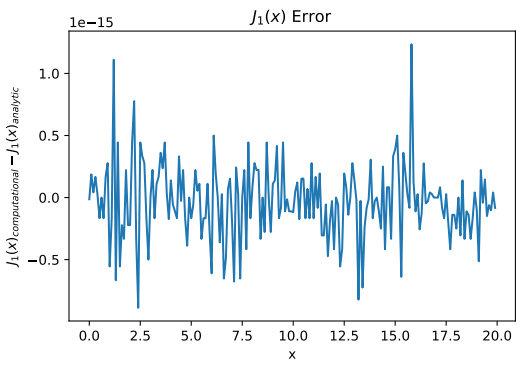


Figure 5: Plotting the error between the computational (Simpson's rule) and analytic (scipy.special.jv) Bessel function J1(x). The error is on the order of 10-15, quite impressive.

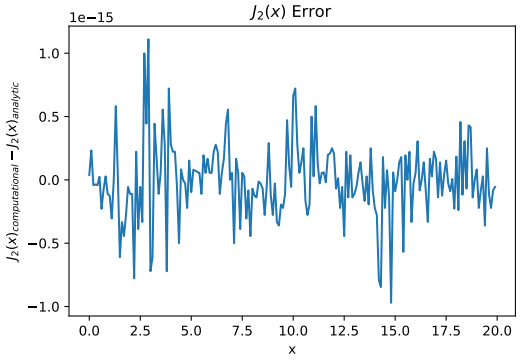


Figure 6: Plotting the error between the computational (Simpson's rule) and analytic (scipy.special.jv) Bessel function J2(x). The error is on the order of 10-15, quite impressive.

5.4 (b)

I used units of nanometers in the code for convenience.

I made the intensity function in the code output ¼ when r was really small, since python can run into issues when dividing very small numbers by very small numbers.

When plotting the diffraction pattern, I made vmax=0.01 so the fainter, outer rings would be visible, and made the aspect ratio 1:1 since matplotlib automatically stretches the plot.

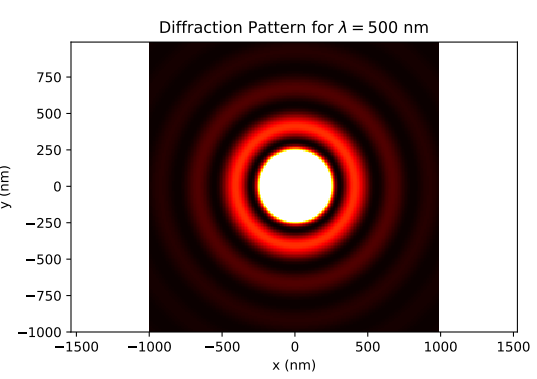


Figure 7: Circular diffraction pattern for λ = 500nm. To make the fainter outer rings appear, I set vmax = 0.01.