ESC407 Lab 1

Maggie Wang

October 2, 2023

1. Diffraction

   (a) The procedure to numerically compute the Bessel function $J_m(x)$ using Simpson's rule is outlined below. fig. 1 shows the results for Bessel functions J_0, J_1, and J_2 as a function of x from $x = 0$ to $x = 2$.

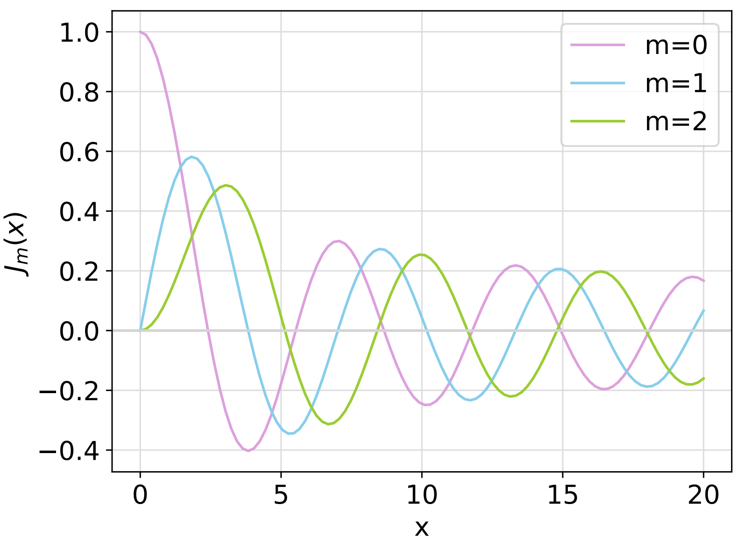

Figure 1: Bessel functions $J_m(x)$ computed using Simpson's rule

Pseudocode

Simpson's rule

```
Define function simpson(N, a, b, f), which computes Simpson's rule on some function f over [a,b] with N slices
    Compute step size h = (b−a)/N
    Initialize sum_f with value f(a)+f(b)
    For each integer k from 1 to N:
            If k is even, add 2f(a+kh) to sum_f
            If k is odd, add 4f(a+kh) to sum_f
        Return h*sum_f/3
```

Bessel function

```
Define function J_integrand(theta, m, x), which returns the integrand of the m^th Bessel function of evaluated at x with integration parameter theta
    Return cos(m theta − x sin(theta))
Define function J(m, x), which returns the m^th Bessel function evaluated at x
    Call simpson with arguments N=1000, a=0, b=π, and f=J_integrand(theta, m = m, x = x)
```

Plotting

```
Initialize and set the number of plotted points nvals=100
For each value of m:
    Initialize x_arr with nvals equally spaced points between 0 and 20
    Compute y_arr as J(m, x) for each x in x_arr
    Plot y_arr vs x_arr
```

(b) The calculation using Simpson's rule closely matches results using `scipy.special.jv`, as seen in fig. 2. The pseudocode to generate fig. 2 is outlined below.

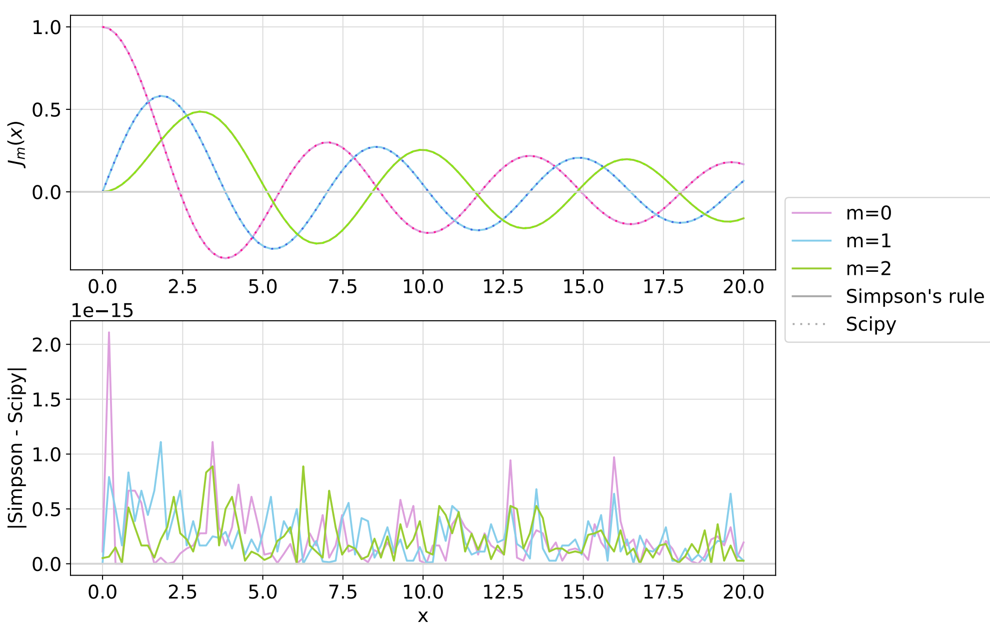

Figure 2: Bessel functions $J_m(x)$ computed using Simpson's rule and `scipy.special.jv`

Pseudocode

```
Initialize and set nvals=100, a=0, b=20
For each value of m of interest:
    Initialize x_arr with nvals equally spaced points from a to b
    Set J_simpson = J(m, x_arr)
    Set J_scipy = scipy.special.jv(m, x_arr)
    Plot J_simpson and J_scipy vs x_arr
```

(c) figure 3 shows a plot of the intensity of the diffraction pattern from a point light source with $\lambda = 500$ nm in the focal plane of a telescope, with Bessel functions computed using Simpson's rule. The pseudocode for the program which generates the plot is outlined below.

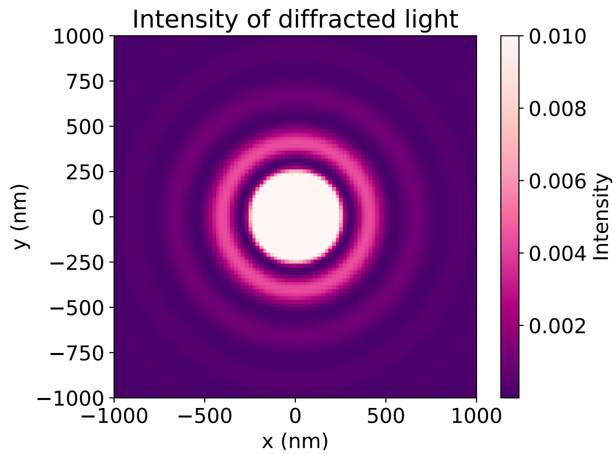

Figure 3: Intensity of diffracted light from 500 nm point source over a radius of 1 um

Pseudocode

```
Define function I(r, wavelength), which returns the intensity of circularly diffracted light at length r from the beam centre and a specified wavelength
    Define k=2π/wavelength
    Call and return (J(1,kr)/(kr))^2
Initialize and set nvals=100, wavelength=500, radius R=1000
Initialize x_array and y_array with num_points evenly spaced values from -R to R
Compute a nvals×nvals array r_mtx with r_ij = x^2 + y^2 for x_i's in x_array and y_j's in y_array
Compute a nvals×nvals array intensity_mtx with entries I(r,wavelength) for each r in r_mtx
```

2. Trapezoidal and Simpson's rules for integration

(a) Using Simpson's rule and Trapezoidal rule with 8 slices, the Dawson function at $x = 4$ is calculated to be `0.1826909645971217` and `0.26224782053479523`, respectively. Using `scipy.special.dawsn`, it is `0.1293480012360051`. Simpson's rule is closer to the value obtained using Scipy, and is more accurate than trapezoidal rule for the same number of slices. However, the relative error using Simpson's rule with 8 slices is still quite large, around 40%. The raw output from the code is as follows:

```
Question 2 a
    Simpson 0.1826909645971217
    Trapezoidal 0.26224782053479523
    Scipy 0.1293480012360051
```

(b) Using Simpson's rule, it takes 1024 slices with a runtime of 0.845 ms (averaged over 50 function calls) to approximate the Dawson function with an error $\mathcal{O}(10^{-9})$, while it takes 65536 slices and

48 ms using trapezoidal rule. In both cases, *scipy.special.dawsn* was used as the reference value. For the same accuracy, in the case where the function is well-behaved, Simpson's rule requires fewer slices and time compared to trapezoidal rule.

The raw output from the code is:

```
Question 2 b)
    Simpson
        value:0.12934800196026494
        N = 1024
        error = 7.242598465406758e-10
        time = 0.001001896858215332 s
    Trapezoidal
        value: 0.12934800371953178
        N = 65536
        error = 2.483526689855964e-09
        time = 0.0678047227859497
```

(c) Using $N_2 = 64$ and $N_1 = 32$, the error estimate of D(4) using Simpson's rule is

$$\epsilon_2 = \frac{1}{15}(I_2 - I_1) \approx 0.00021,$$

and for trapezoidal rule,

$$\epsilon_2 = \frac{1}{3}(I_2 - I_1) \approx 0.00051$$

The raw output from the code is:

```
Question 2 c)
    Simpson: 0.00020578842293380212
    Trapezoidal: 0.0005093137305911358
```

3. Exploring roundoff error

(a) fig. 4 shows $p(u) = (1-x)^8$ and $q(u)$, the Taylor expansion of $p(u)$ up to degree 8. The plot of $q(u)$ appears noisier because there are more terms involved, each affected by roundoff error.

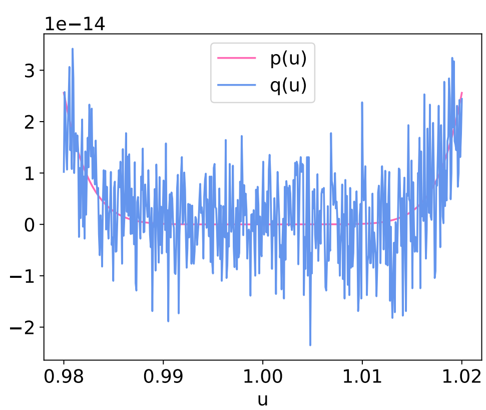

Figure 4: $p(u) = (1-x)^8$ and its Taylor expansion to degree 8, $q(u)$, around u = 1

(b) fig. 5 a) plots $|p(u) - q(u)|$ around u = 1. fig. 5 b) shows the histogram associated with fig. 5 a), which has a standard deviation of 8×10^{-15}. This standard deviation should be the same order of magnitude as an estimate obtained using equation 1, since they are both measures of roundoff error when summing over multiple terms.

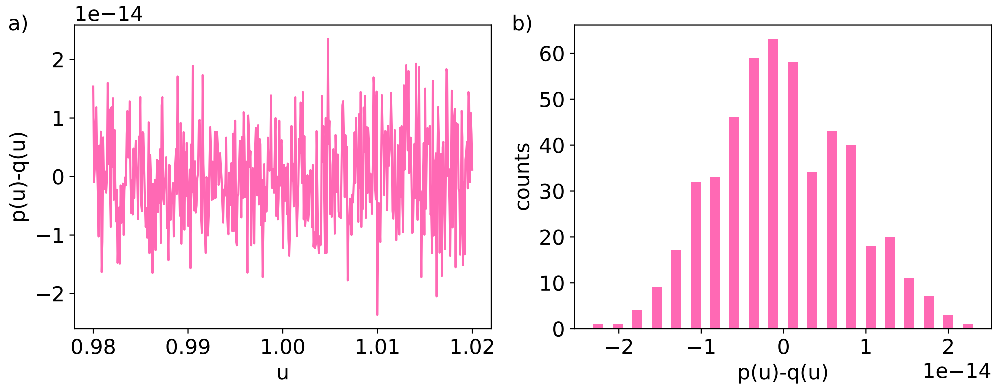

Figure 5: a) $|p(u) - q(u)|$ around u = 1 and b) the corresponding histogram

The estimate of the error obtained using equation 1 for $p(1) - q(1)$ is the same order of magnitude as the standard deviation above.

$$\begin{aligned}\sigma &= C\sqrt{N}\sqrt{\overline{x^2}} \\ &= 10^{-16}\sqrt{10}\sqrt{1287} \\ &= 1.1 \times 10^{-14}\end{aligned}$$

The operation $p(u) - q(u)$ is a sum over $p(u)$ and each term in $q(u)$, for $u = 1$. Since there are 10 terms in total, $N = 10$. $\overline{x^2}$ is calculated as

$$\begin{aligned}\overline{x^2} &= \frac{(1-1)^{8\cdot 2} + 1^2 + 8^2 + 28^2 + 56^2 + 70^2 + 56^2 + 28^2 + 8^2 + 1^2}{10} \\ &= 1287\end{aligned}$$

The direct output from the code is

```
3b:  std of p(u)-q(u) 7.992292568650469e-15
```

(c) Equation 2 estimates the fractional error as

$$\frac{\sigma}{\sum_i x_i} = \frac{C}{\sqrt{N}}\frac{\sqrt{\overline{x^2}}}{\overline{x}}$$

In the calculation of $p(u) - q(u)$, the x_i's alternates sign and as a result, $\overline{x}$ is much smaller than $\sqrt{\overline{x^2}}$ and the fractional error becomes quite large. Using $N = 10$ and the same method to calculate $\overline{x^2}$ as above, for $u = 0.985$,

$$\begin{aligned}\sqrt{\overline{x^2}} &= 33.778 \\ \overline{x} &= 2.232 \times 10^{-15} \\ \frac{\sigma}{\sum_i x_i} &= 0.478\end{aligned}$$

For $u = 0.99$,

$$\sqrt{\overline{x^2}} \approx 34.465$$
$$|\overline{x}| = 1.898 \times 10^{-16}$$
$$|\frac{\sigma}{\sum_i x_i}| = 5.741$$

Between $u = 0.985$ and $u = 0.99$, the estimated fractional error approaches 1.00
fig. 6 shows that the relative error $|p(u) - q(u)|/|p(u)|$ approaches 1.00 when u is between 0.978 and 0.985

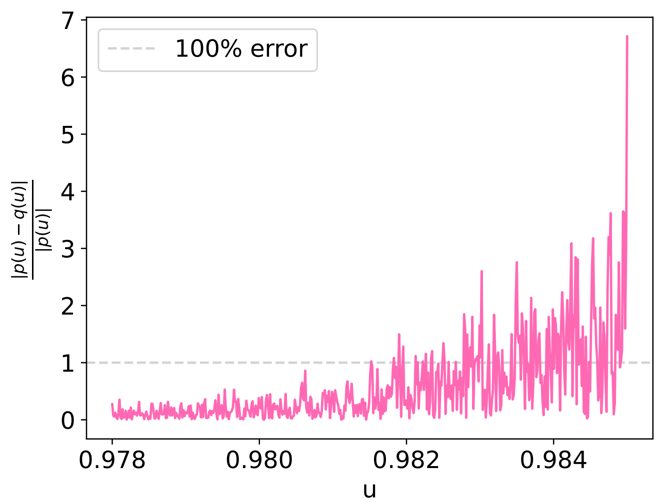

Figure 6: Relative error $|p(u) - q(u)|/p(u)$ for u between 0.978 and 0.985

(d) Plotting $u^8/(u^4u^4) - 1$ in fig. 7 shows the error is on the order of 10^{-16}, which is the same as what equation (4.5), $\sigma = \sqrt{2}Cx = 1.41 \times 10^{-16}$, predicts for $C = 10^{-16}$ and $x = 1$

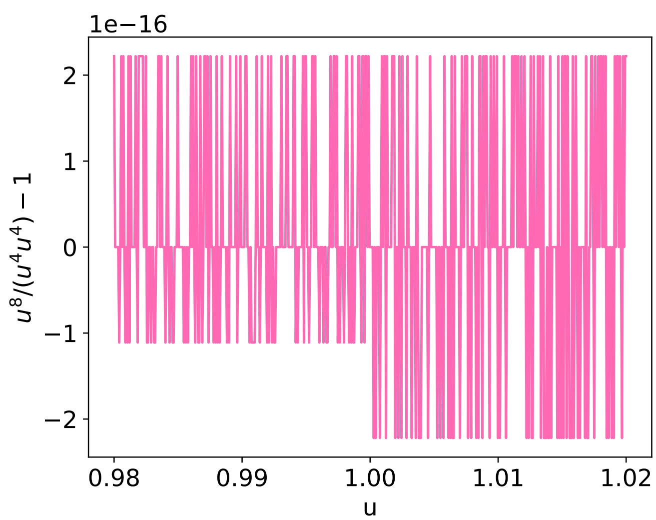

Figure 7: $u^8/(u^4u^4) - 1$ around u=1