Lab assignment #2: Numerical Errors — Numerical Integration
Solution

Nicolas Grisouard (Instructor), Mikhail Schee (TA & Marker), Alex Cabaj (TA)

Due Friday, September 25th 2020, 5 pm

Note: all of these problems re-use the same set of integration routines in `intfuncs.py`

1. **[20% of the lab] Numerical differentiation errors:**

   See `L02-407-2020-Q1.py` for the code.

   (a) Nothing to submit.

   (b) Here is the output from the code:

```
   h      |    fwd    |  cntrd    | fwd err  | cntrd err
1.00e+00 | -6.73e-01 | -3.37e-01 | 1.35e-01 | 5.68e-01
1.00e-01 | -8.11e-01 | -7.72e-01 | 4.17e-02 | 8.30e-03
1.00e-02 | -7.83e-01 | -7.79e-01 | 4.92e-03 | 8.33e-05
1.00e-03 | -7.79e-01 | -7.79e-01 | 4.99e-04 | 8.33e-07
1.00e-04 | -7.79e-01 | -7.79e-01 | 5.00e-05 | 8.33e-09
1.00e-05 | -7.79e-01 | -7.79e-01 | 5.00e-06 | 8.87e-11
1.00e-06 | -7.79e-01 | -7.79e-01 | 5.00e-07 | 5.31e-11
1.00e-07 | -7.79e-01 | -7.79e-01 | 4.95e-08 | 4.09e-10
1.00e-08 | -7.79e-01 | -7.79e-01 | 9.57e-09 | 2.44e-09
1.00e-09 | -7.79e-01 | -7.79e-01 | 2.38e-08 | 2.38e-08
1.00e-10 | -7.79e-01 | -7.79e-01 | 8.79e-07 | 1.66e-07
1.00e-11 | -7.79e-01 | -7.79e-01 | 1.97e-06 | 1.97e-06
1.00e-12 | -7.79e-01 | -7.79e-01 | 2.65e-05 | 2.65e-05
1.00e-13 | -7.79e-01 | -7.79e-01 | 7.39e-04 | 2.65e-05
1.00e-14 | -7.77e-01 | -7.77e-01 | 2.11e-03 | 2.11e-03
1.00e-15 | -7.77e-01 | -7.77e-01 | 2.11e-03 | 2.11e-03
1.00e-16 | -1.11e+00 | -1.11e+00 | 4.26e-01 | 4.26e-01
```

   (c) The printed output (column `fwd err`) shows that indeed, the smallest error occurs for $h_m = 10^{-8}$ (see fig. 1, solid line, for the plot).

   For $h < h_m$, the error goes up because of round-off error: we subtract numbers that are increasingly close together, so close that their errors add up to a number that is relatively larger than the difference. From eqn. (5.91) in the text, this error goes as h^{-1}, or, in log-log coordinates, $\log\epsilon \sim -\log h$. Indeed, we will see on fig. 1 that the decay is on a -1 slope in log-log.

   For $h > h_m$, the error goes up because of approximation (or truncation) error: a difference is only so good an approximation of a derivative, and as the points in the difference spread apart, the approximation gets worse. From eqn. (5.91) in the text, this error goes as h, or, in log-log coordinates, $\log\epsilon \sim \log h$. Indeed, we will see on fig. 1 that the decay is on a +1 slope in log-log.

(d) The forward difference error reflects the fact that the truncation error goes as h^2, as described by eqn. (5.99): larger steps are associated with smaller errors, and the trucation error goes as $\log\epsilon \sim 2\log h$, as visible on the figure.

The round-off error is more or less the same for $h < 10^{-8}$, though dominates for $10^{-8} < h < 10^{-6}$. The optimal step is now 10^{-6} form the figure, disagreeing with the text (10^{-5}, discussion below eqn. 5.10), though not by much, especially as the text is explicit about it being only an estimate. Finally, while the optimal time step is larger, the error is much smaller, (10^{-10} vs. 10^{-8},same as what the textbook predicts).

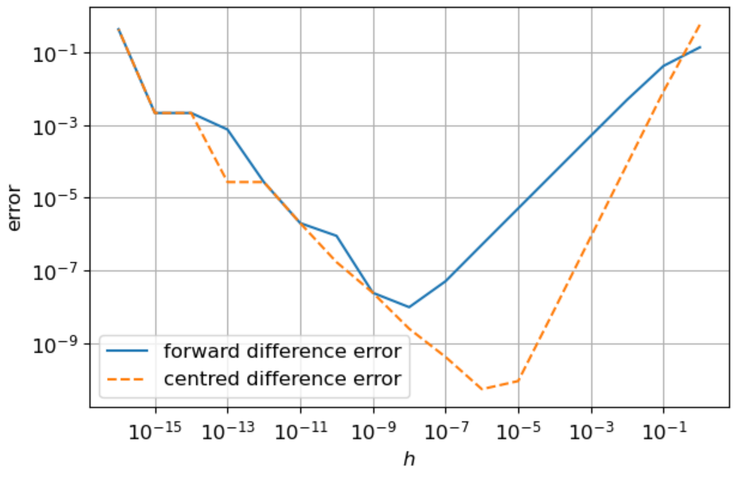

Figure 1: Plots of the errors as a function of step size.

2. **[40% of the lab] Trapezoidal and Simpson's Rules for Integration.**

(a) Dawson function.

See `L02-407-2020-Q2a.py` for the code.

i. Test with $N = 8$:

```
According to scipy,        D(4.0) = 0.1293480012360051

Test with N=8 slices
With the Trapezoidal rule, D(4.0) = 0.2622478205347951
With Simpson's rule,       D(4.0) = 0.18269096459712167
```

At this point, the Trapezoidal rule agrees with SciPy... Somewhat. Let's say that the order of magnitude is correct. Simpson's rule on the other hand agrees with it up to the... 1st significant digit. Let's say that it is a bit better. Note that the questions below will allow us to be more confident in how well the integrals estimate the value.

ii. Increasing N by powers of two each time:

```
Simpson, with N = 8: error = 0.053342963361116574
Simpson, with N = 16: error = 0.007618483860179381
Simpson, with N = 32: error = 0.000663180347746789
Simpson, with N = 64: error = 4.5815078945382615e-05
... Skipping 2 more values of N ...
Simpson, with N = 512: error = 1.1583255937752668e-08
Simpson, with N = 1024: error = 7.242598742962514e-10

Trapezoidal, with N = 8: error = 0.13289981929879002
Trapezoidal, with N = 16: error = 0.03893881771983207
Trapezoidal, with N = 32: error = 0.01023208969076811
Trapezoidal, with N = 64: error = 0.0025923837319010434
... Skipping 9 more values of N ...
Trapezoidal, with N = 65536: error = 2.483527022922871e-09
Trapezoidal, with N = 131072: error = 6.208822622699728e-10
```

Note: wrapping everything into functions, then using the 'while' statement, are somewhat overkills. We accepted less elegant ways to do it (increasing the number of slices "by hand", etc.), but this way of doing is more systematic and modular, therefore more readable and easily de-bugged.

- Simpson's rule hits the 10^{-10} mark with $N = 1024$ slices.
- The trapezoidal rule hits it with $N = 131072 = 2^{17}$ slices.

Additional remarks:

- For Simpson's rule, notice how the relative deviation from the "true" result gets divided by ≈ 16 every time the number of slices gets multiplied by 2 (at least near the end). It is the hallmark of a 3rd-order accurate integration method ($O(h^4)$ error).
- On the other hand, for the trapezoidal rule, the relative deviation from the "true" result "only" gets divided by ≈ 4 every time the number of slices gets multiplied by 2, characteristic of a 1st-order of accuracy ($O(h^2)$ error).

As for the timing, here is what the routine produces, after executing each integration 100 times:

```
Integrating with an error of 1e-09 with Simpson's rule takes 1.805e-03 s.
Integrating with an error of 1e-09 with trapezoidal rule takes 2.485e-01 s.
It takes 2.328e-06 s for scipy to compute D(4.0).
```

So, for the same level of accuracy, Simpson's rule is about 130 times faster. And using SciPy is really the way to go.

iii. We go back to Q2a.ii, go from $N = 32$ to $N = 64$, and use eqns. (5.26) to (5.29) of the textbook to compute the error estimates for both methods.

```
Practical estimation of epsilon for trapz =  -0.0025465686529556886
Practical estimation of epsilon for Simpson =  -4.115768458676043e-05
```

Note that in absolute value, these estimates are pretty spot on, when looking at the errors reported in Q2a.ii for $N = 64$. Usually though, this is a leading-order kind of estimate, especially with a small number of slices.

(b) Diffraction limit of a telescope (Exercise 5.4, p. 148 of the textbook).

See `L02-407-2020-Q2b.py` for the code.

- Computing the Bessel functions: see fig. 2, left panel.

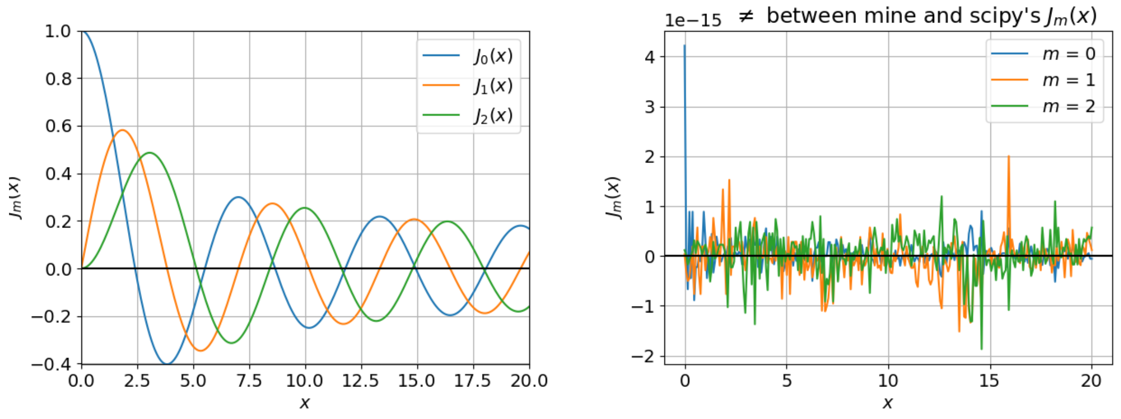

Figure 2: Left: first three orders for the Bessel functions, computed with a Simpson integration of our own. Right: differences between my own Bessel functions and the ones from `scipy.special.jv`.

- Difference with `scipy.special.jv`: see fig. 2, right panel. Note the `1e-15` multiplier on the y-axis: not too shabby!
- Polar plot: I believe that the question asks you to convert (r, θ) to (x, y). I will do it below, but first, I show a "native polar plot" on fig. 3, left panel.

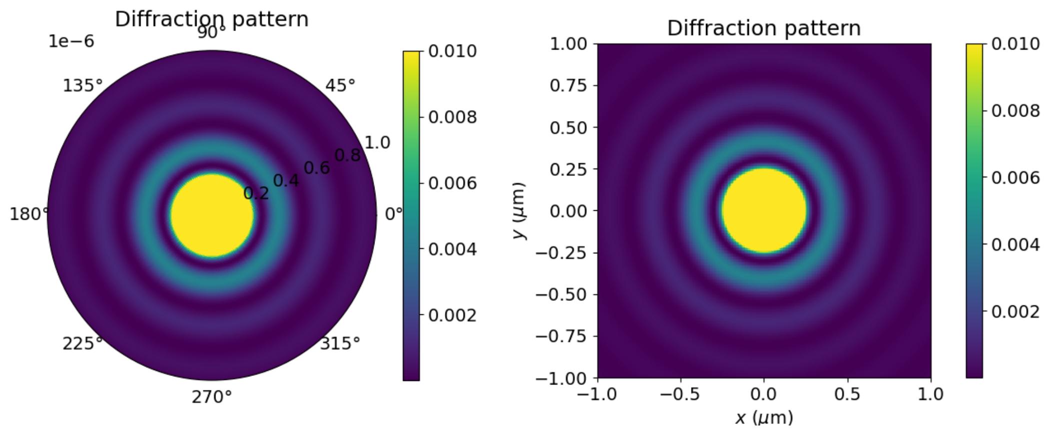

Figure 3: Diffraction spots, in (left) "native polar coordinates" and (right) Cartesian coordinates. Either plots worked to get all points.

The right panel of the same figure show the same plot, with coordinates converted to Cartesian.

3. **[40% of the lab] Diffraction gratings**

   (a) Nothing to submit.

   (b) Nothing to submit.

   (c) We are first asked to write a function that will return the amplitude of the grating. We will use the Simpson's rule, because (i) we already coded it and (ii) is is almost always better than the trapezoidal rule, which are the two rules we know so far. We have 10 slits to integrate over, meaning we need a reasonable number of slices in each slit. Note that the Euler-MacLaurin formula fials in this case: the array of slits is periodic, meaning that $f'''(a) - f'''(b) = 0$, where a and b are the integration bounds! We therefore settled on 100 slices per slits, or 1000 slices total, because the function is relatively smooth on this interval and we know that the Simpson formula will give excellent results with so many slices.

The pseudocode to carry out this program would look something like

```
# import required packages, including plotting
# Define constants for grating

# define functions q and ig

# set limits of integration, and other parameters
# a, b = -w/2, w/2
# h = ...
# N = ...
# calculate the integral using Simpson's rule
# Make the density plot.
```

No need to go into too much detail.

(d) The plot for the $\sin^2(\alpha u)$ grading with 10 slits is in fig. 4.

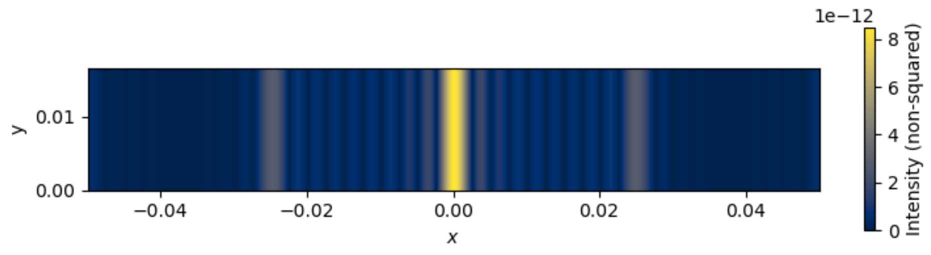

Figure 4: Diffraction pattern for a grating $q(u) = \sin^2(\alpha u)$ with 10 slits. Note that compared with the eqn. at the bottom of p. 206, we are plotting $\sqrt{I}$. On the other hand, our figure seems to match that of p. 207.

Not much to comment about actually: it is what it is! A central band of increased intensity, two flanking bands, with more interference patterns in-between.

(e) The two plots are not in figs 5 (grating i) and 6 (grating ii).

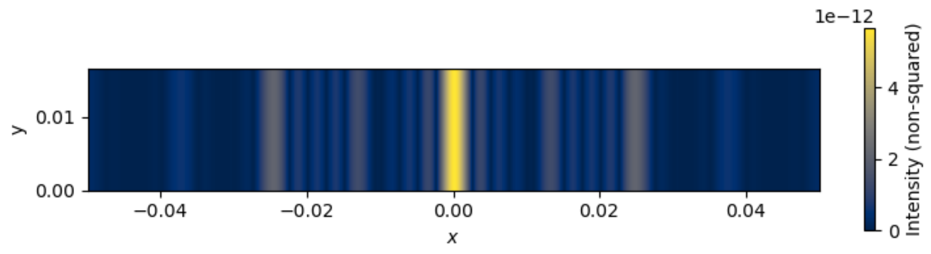

Figure 5: Diffraction pattern for a grating of two open slits of unequal widths. Note that like in fig. 4, we are plotting $\sqrt{I}$.

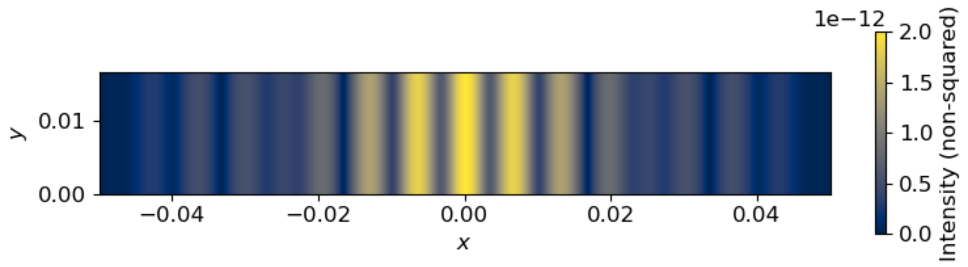

Figure 6: Diffraction pattern for a grating $q(u) = \sin^2(\alpha u)$ with 10 slits. Note that like in figs. 4 and 5, we are plotting $\sqrt{I}$.