Lab assignment #2: Numerical Errors — Numerical Integration

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)

Due Friday, September 25th 2020, 5 pm

We are still trying to keep the gather.town system for now, but the information below may change (watch announcements on Quercus).

Room 1 (groups #1–20) Mikhail Schee (Marker, mikhail.schee@mail.utoronto.ca),
URL: `https://gather.town/ZdDgdR4j2gGZIqvc/MS-PHY407`
PWD: `phy407-2020-MS`

Room 2 (groups #21–40) Alex Cabaj (alex.cabaj@mail.utoronto.ca),
URL: `https://gather.town/2d25ninrKWziTcd4/PHY407AC`
PWD: `phy407-2020-TA-Alex`

Room 3 (groups #>40) Nicolas Grisouard (nicolas.grisouard@utoronto.ca),
URL: `https://gather.town/SluBq0pSNaWQycc4/phy407-NG`
PWD: `phy407-NG`

General Advice

- **Work with a partner!**
- Read this document and do its suggested readings to help with the pre-labs and labs.
- The goals of this lab are to put into practice knowledge about numerical errors, and the trapezoid and Simpson's rules for integration.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Specific instructions regarding what to hand out are written for each question in the form:

  THIS IS WHAT IS REQUIRED IN THE QUESTION.

  Not all questions require a submission: some are only here to help you. When we do though, we are looking for "C^3" solutions, i.e., solutions that are **C**omplete, **C**lear and **C**oncise.
- An example of **C**larity: make sure to label all of your plot axes and include legends if you have more than one curve on a plot. Use fonts that are large enough. For example, when

integrated into your report, the font size on your plots should visually be similar to, or larger than, the font size of the text in your report.

- Whether or not you are asked to hand in pseudocode, you **need** to strategize and pseudocode **before** you start coding. Writing code should be your last step, not your first step. Test your code as you go, **not** when it is finished. The easiest way to test code is with `print()` statements. Print out values that you set or calculate to make sure they are what you think they are. Practice modularity. It is the concept of breaking up your code into pieces that as independent as possible form each other. That way, if anything goes wrong, you can test each piece independently. One way to practice modularity is to define external functions for repetitive tasks. An external function is a piece of code that looks like this:

```
def MyFunc(argument):
    """A header that  explains the function
    INPUT:
    argument [float] is the angle in rad
    OUTPUT:
    res [float] is twice the argument"""
    res = 2.*argument
    return res
```

  For example, in this lab, you will probably use Trapezoidal and Simpson's rules more than once. You may want to write generic functions for these rules (or use the piece of code, provided by the textbook online resources), place them in a separate file called e.g. `functions_lab02.py`, and call and use them in your answer files with:

```
import functions_lab02 as fl2  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl2.MyFunc(ZehValyou)
```

Computational background

Forward and Centred Differences Anticipating on the week about differentiating functions, we will use forward and centred approximations of a function, i.e.,

$$\text{forward difference:} \quad \left.\frac{df}{dx}\right|_{x_i} \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \quad \text{and} \tag{1}$$

$$\text{centred difference:} \quad \left.\frac{df}{dx}\right|_{x_i} \approx \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}}. \tag{2}$$

Solving integrals numerically Lecture notes, as well as Sections 5.1 – 5.3 of the text, introduce the Trapezoidal Rule and Simpson's Rule. The online resource for the text provides the python program `trapezoidal.py`, which you are free to use.

Complex numbers Python uses `1j` to represent the imaginary i number, but it doesn't stop there. As an example, two ways to represent the complex number $z = 1 + 2i$, with $i^2 = -1$ with NumPy could be

```
z = 1 + 2j
```

and

```
z = numpy.complex(1., 2.)
```

It also works with e.g. $z = 1 + i/2$, which is

```
z = 1 + 0.5j
```

(though `z=1+j/2` does not work, but `z=1+1j/2` does) and

```
z = numpy.complex(1., 0.5)
```

SciPy special functions You will be asked to compute functions, defined based on integrals (e.g., Bessel functions). You will also be asked to compare with pre-coded versions of the same functions. These are a little too exotic to be part of NumPy, but common enough to be part of `scipy.special`. Import them at the beginning of your code (`from scipy.special import XX`, with `XX` the function you want to use), and use as `XX(value)`. Here they are:

- m^{th}-order Bessel function of the first kind $J_m(x)$: `scipy.special.jv`,
- Dawson's integral $D(x)$: `scipy.special.dawsn`

Anonymous functions Sometimes, you need a function without a name. For example, you want to sum the two values of the function $f(x,z)$ at a and b,

$$S(z) = f(a,z) + f(b,z).$$

The function f has a name ($f\ldots$), but what you *really* want is to treat z as a constant, and pretend $f(x,z)$ is really a function of one variable, x, but could not be bothered to define another Python function to do just that. Easy! Just use an anonymous function thanks to `lambda`. For example,

```
import numpy as np
def f(x, z):
    """The function with a name"""
    return np.sin(x*z)

def la_sum(a, b, func):
    """ Python function that add two values of func from a to b """
    return func(a) + func(b)

# You can't just do la_sum(a, b, f), because f takes two arguments.
# Enters an anonymous function
S = la_sum(a, b, lambda x: f(x, z))
# in the example above, z is just a fixed parameter for the anonymous
# function of x that we sum.
```

Choosing a colour map on density plots. This is an art as much as it is a science, and we will not be strict about it in this lab. However, the conversation has evolved significantly since Newman wrote his book, and his suggestions might not be the best. You can check the following article out:

> Zeller, S., and D. Rogers (2020), Visualizing science: How color determines what we see, Eos, 101, https://doi.org/10.1029/2020EO144330. Published on 21 May 2020.

Questions

1. **[15% of the lab] Numerical differentiation errors:** Read section 5.10.2 in the text. Demonstrate that the optimum step size for forward difference differentiation schemes is indeed $\sqrt{C} \approx 10^{-8}$ by using the following example.

   (a) Consider the function $f(x) = e^{-x^2}$. Using a forward difference scheme, numerically find the function's derivative at $x = 0.5$ for a range of h's from $10^{-16} \rightarrow 10^{0}$ increasing by a factor of 10 each step (so you should have 17 values for h with respective values of the derivative for each h value).

   NOTHING TO SUBMIT (YET).

   (b) Calculate the error in each derivative by comparing the value of the numerical derivative that you get to the analytic value. Take the absolute value of the error.

   INCLUDE THE VALUES YOU FIND FOR PARTS (A) AND (B, ABSOLUTE VALUES ONLY), SIDE-BY-SIDE.

   (c) Plot the error as a function of step size on a log-log plot and show that the minimum is indeed at $\approx 10^{-8}$. Explain the shape of the curve by considering equation (5.91) in the text. When does the truncation error dominate? When does the rounding error dominate?

   SUBMIT THE WRITTEN ANSWERS TO THESE QUESTIONS.

   (d) Now implement a central difference scheme for the derivative and calculate the error for the same function. Plot the error in the central difference scheme on the same plot as your error for the forward difference scheme from part (c). Concisely explain the key features of your plot. Does the central difference scheme ALWAYS clearly beat the forward difference scheme in terms of accuracy?

   SUBMIT YOUR CODE AND INCLUDE THE PLOT, WITH BOTH CURVES ON IT, AND ANSWERS TO THE QUESTIONS.

2. **[50% of the lab] Trapezoidal and Simpson's Rules for Integration.**

   (a) We seek to evaluate the Dawson function

   $$D(x) = \mathrm{e}^{-x^2} \int_0^x \mathrm{e}^{t^2} \mathrm{d}t \tag{3}$$

   at $x = 4$.

   i. For $N = 8$ slices, compare the value you obtain when using the Trapezoidal vs. Simpson's rule, and with the value given by `scipy.special.dawsn`.

   ii. For each method (Trapezoidal and Simpson), how many slices do you need to approximate the integral with an error of $O(10^{-9})$? Use SciPy's value as the "true" value. We are only looking for a rough estimate for the number of slices for each method. I.e., $N = 2^n$, with $n = 3, 4, 5 \ldots$, and N or n being the answer. For this question, do it in a "dumb" way: increase the number of slices until you hit the mark.

How long does it takes to compute the integral with an error of $O(10^{-9})$ for each method (including SciPy's version)?

Note: the integration is fast in both methods. In order to get an accurate timing, you may want to repeat the same integration many times over. Recall that we described a timing method in last week's lab, but you are free to choose functions with better performance.

iii. Adapt the "practical estimation of errors" of the textbook (§ 5.2.1, p. 153) to both methods to obtain the error estimation for $N_2 = 64$ (using $N_1 = 32$).

FOR THE WHOLE EXERCISE ON THE DAWSON FUNCTION, SUBMIT YOUR CODE (IN A SEPARATE FILE FROM Q2B), PRINTED OUTPUTS, AND WRITTEN ANSWERS.

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

- Do Exercise 5.4(a).
- Then, compare graphically the difference between your Bessel functions and those from `scipy.special.jv`. How well do you reproduce the SciPy routines with your own Bessel function?
- Now do Exercise 5.4(b). Read the hints, but if you have some extra time, I recommend looking into

  `matplotlib.pyplot.pcolormesh`

  instead of

  `matplotlib.pyplot.imshow`,

  though the latter will do for our purposes.

FOR THE WHOLE EXERCISE ON DIFFRACTION LIMIT OF A TELESCOPE, SUBMIT YOUR CODE (IN A SEPARATE FILE FROM Q2A), PLOTS, PRINTED OUTPUTS, AND WRITTEN ANSWERS.

3. **[35% of the lab] Diffraction gratings** Exercise 5.19, p. 206 of the textbook.

(a) Skip 5.19(a). *The slits will occur at the zeros of the function, i.e., $\alpha z = n\pi$, with $n \in \mathbb{Z}$. Therefore, the distance between adjacent slits is π/α.*

(b) Do 5.19(b).

NOTHING TO SUBMIT.

(c) Do 5.19(c).

Note: You may ignore the comments about the `cmath` package, and simply use `numpy.exp`, which accepts complex arguments.

SUBMIT YOUR PSEUDOCODE, SHORT EXPLANATIONS OF HOW YOU MADE YOUR CHOICES, AND WHICH CRITERIA YOU BASED THEM ON.

(d) Do 5.19(d).

Do not submit such a bare figure as the one on p. 207. Indeed, we expect labels and ticks at least on the horizontal axis, and a labelled colour bar

Note: I suspect that the figure on p. 207 is incorrect: the author might have forgotten to take the square of $|\int(\ldots)du|$ *at the bottom of p. 206. Unless it is said formula that is wrong. In any case, omitting the square makes the figure nicer to read, actually. So, submit whichever plot you prefer, as long as you clearly explain what you are submitting.*

SUBMIT YOUR PLOT.

(e) Do 5.19(e).

SUBMIT YOUR CODE, WHICH AT THIS POINT SHOULD INCLUDE ALL OF THE DIFFERENT GRATINGS MENTIONED IN 5.19(A–E), YOUR FINAL PLOTS, AND COMMENTS ABOUT (OR DESCRIPTIONS OF) THE FIGURES.