Lab assignment #2: Numerical Errors and Numerical Integration

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)
TA & Marker: Alex Cabaj (alex.cabaj@mail.utoronto.ca)

Due Friday, September 24th 2021, 5 pm

Each room has a capacity of 25 participants. First, try to sign up for room 1. If it is full, sign up for room 2 and ask your partner to join you there if they are in room 1. And so on until you and your partner find a room.

Room 1 (also for office hour) Alex Cabaj,
https://gather.town/app/vmCKPZTSEKgGYlZI/PHY407AC1
PWD: `tr0p0sph3r3`

Room 2 Nicolas Grisouard, https://gather.town/app/2iGkhRu89WVOdW4X/phy407-ng-room1
PWD: `phy407-ng-room1`

Room 3 Alex Cabaj, https://gather.town/app/fgHzcNuj6SI9CqNu/PHY407AC2
PWD: `str4t0sph3r3`

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

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 textbook provides the Python program `trapezoidal.py`, which you are free to use.

Standard deviation calculations To calculate the sample mean $\overline{x}$ and standard deviation σ of a sequence $\{x_1, \ldots, x_n\}$ using the standard formulas

$$\overline{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \sigma \equiv \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2} \tag{1}$$

requires two passes through the data: the first to calculate the mean and the second to compute the standard deviation (by subtracting the mean in the term $(x_i - \overline{x})$ before squaring it). An alternative, mathematically equivalent, formula for the standard deviation,

$$\sigma \equiv \sqrt{\frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - n\overline{x}^2\right)}, \tag{2}$$

might seem preferable because, if you think about it, you can calculate σ in eqn. (2) with only a single pass through the data. This means, for example, that you could calculate these statistics on

a live incoming data stream as it updates. However, the one-pass method has numerical issues that you will investigate in Q1.

In Q1 we will compare both methods to the canned routine `numpy.std(array, ddof=1)`, which also implements eqn. (1), and which you can think of as a correct calculation.

Reading data from a textfile The command

```
a = numpy.loadtxt('filename.txt')
```

will read data in the file `'filename.txt'` into the array `a`.

SciPy special functions You will be asked to compute functions, defined based on integrals (in our case, 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)`.

- For the m^{th}-order Bessel function of the first kind $J_m(x)$, replace `XX` with `jv`, and use it as `jv(m, x)`.
- For the m^{th}-order modified Bessel function of the second kind $K_m(x)$, replace `XX` with `kn`, and use it as `kn(m, x)`.

See the full list at

https://docs.scipy.org/doc/scipy/reference/special.html

SciPy constants The `SciPy` package includes some of the most common constants in Physics and other scientific fields:

https://docs.scipy.org/doc/scipy/reference/constants.html

Making a three dimensional surface plot Matplotlib provides the limited `mplot3d` package to represent three dimensional surfaces on your screen or on the page. A tutorial can be found at

https://matplotlib.org/stable/tutorials/toolkits/mplot3d.html#surface-plots

For Q2, you can adapt the code in `surface3d.py` (available at the above URL if you click on the picture) to create the requested plot.

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. Nonetheless, 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.

If you feel like installing extra modules, Mikhail Schee introduced me to a nice set of carefully-crafted colour maps: https://colorcet.holoviz.org

Physics background

Vibrating membrane The wave equation for a vibrating membrane like a drumhead with shape $u(\vec{x},t)$, where $\vec{x}$ is the horizontal position, is given by

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u, \tag{3}$$

where c is the wave phase speed which is controlled by the tension of the membrane. Consider a circular membrane, like a drum, of radius R, subject to the boundary condition that the membrane is fixed at a distance R from the origin. In polar coordinates (r,θ), this boundary conditions is written $u(r=R,\theta,t)=0$. Then using separation of variables the general solution to the wave equation is given (neglecting phase and multiplicative constants) by

$$u_{mn} = J_n\left(\frac{z_{mn} r}{R}\right)\cos(n\theta)\cos\left(\frac{c z_{mn} t}{a}\right). \tag{4}$$

In (4), J_n is the order-n Bessel function of the first kind, whose integral form is

$$J_n(x) = \frac{1}{\pi}\int_0^\pi \cos(n\phi - x\sin\phi)\,\mathrm{d}\phi. \tag{5}$$

In addition, in (4), z_{mn} is the m^{th} zero of the order-n Bessel function of the first kind, i.e. z_{mn} satisfies $J_n(z_{mn})=0$ for each m, n.

Electrostatic potential for a line of charge Consider an infinite line of charge along the z axis with most of the charge concentrated near the origin. The charge per unit length is

$$\lambda(z) = \frac{Q}{\ell}\mathrm{e}^{-z^2/\ell^2}, \tag{6}$$

where Q and ℓ are constants.

At the position (r,z), where r is the radial coordinate, the electrostatic potential is given by the integral

$$V(r,z) = \int_{-\infty}^{\infty}\frac{\lambda(z')\,\mathrm{d}z'}{4\pi\epsilon_0\sqrt{(z-z')^2+r^2}} = \int_{-\infty}^{\infty}\frac{Q\mathrm{e}^{-z'^2/\ell^2}\,\mathrm{d}z'}{4\pi\epsilon_0\ell\sqrt{(z-z')^2+r^2}} \quad \text{for} \quad r>0, \tag{7}$$

where the integration accounts for all contributions of the infinite line of charge to to the potential at (r,z). This integral is well behaved away from $r=0$. To calculate it numerically, it is useful to change coordinates in the integrand as suggested by the equation above: the transformation to use is $z' = \ell\tan u$ and this results in

$$V(r,z) = \int_{-\pi/2}^{\pi/2}\frac{Q\mathrm{e}^{-(\tan u)^2}\,\mathrm{d}u}{4\pi\epsilon_0\cos^2 u\sqrt{(z-\ell\tan u)^2+r^2}}. \tag{8}$$

For $z=0$, a known result is that eqn. (7) becomes

$$V(r,z=0) = \frac{Q}{4\pi\epsilon_0\ell}\mathrm{e}^{r^2/2\ell^2}K_0\left(r^2/2\ell^2\right), \tag{9}$$

where $K_0(x)$ is the zeroth-order modified Bessel function of the second kind (see computational background). Comparing your solution to this case will be useful to determine the accuracy of your integral calculation.

Questions

1. **[35%] Exploring numerical issues with standard deviation calculations**

   (a) Write pseudocode to test the relative error found when you estimate the standard deviation using the two formulas (1) and (2), treating the `numpy` method

   ```
   numpy.std(array, ddof=1)
   ```

   as the correct answer. The input for this calculation will be a supplied dataset consisting of a one-dimensional array of values that is read in using `numpy.loadtxt()`.

   Note: The relative error of a value x compared to some true value y is $(x-y)/y$.

   In implementing (2), you will need to account for the possibility that this approach could result in taking the square root of a negative number. Why is this check necessary? You can implement a stopping condition if this is the case, or print a warning.

   SUBMIT THE PSEUDO-CODE.

   (b) Now, using this pseudocode, write a program that uses (1) to calculate the standard deviation of Michelsen's speed of light data (in $10^3\,\mathrm{km\,s^{-1}}$), which is stored in the file `cdata.txt`, and which was taken from John Baez's (U.C. Riverside) website.[1]

   Calculate the relative error with respect to `numpy.std(array, ddof=1)`. Now do the same for eqn. (2). Which relative error is larger in magnitude?

   SUBMIT THE CODE, PRINTED OUTPUT, AND WRITTEN ANSWERS TO QUESTIONS.

   (c) To explore this question further, we will evaluate the standard deviation of a sequence with a predetermined sample variance. The function

   ```
   numpy.random.normal(mean, sigma, n)
   ```

   returns a sequence of length `n` of values drawn randomly from a normal distribution with mean `mean` and standard deviation `sigma`. Now generate two normally distributed sequences, one with

   ```
   mean, sigma, n =  (0., 1., 2000)
   ```

   and another one with

   ```
   mean, sigma, n = (1.e7, 1., 2000),
   ```

   with the same standard deviation but a larger mean. Then evaluate the relative error of (1) and (2), compared to the `numpy.std` call. How does the relative error behave for the two sequences?

   Now that you have investigated a few cases, can you explain the difference in the errors in the two methods, both for these distributions and for the data in Q1b?

   SUBMIT PRINTED OUTPUT AND WRITTEN ANSWERS TO QUESTIONS.

   (d) Can you think of a simple workaround for the problems with the one-pass method encountered here? Try this workaround and see if it fixes the problem.

[1] http://math.ucr.edu/home/baez/physics/Relativity/SpeedOfLight/measure_c.html

SUBMIT PRINTED OUTPUT AND WRITTEN ANSWERS TO QUESTIONS.

2. **[45%] Trapezoidal and Simpson's rules for integration.**

   (a) We seek to evaluate the integral

   $$\int_0^1 \frac{4}{1+x^2}\,\mathrm{d}x. \tag{10}$$

   i. What is the exact value of this integral?

   ii. For $N = 4$ slices, compare the value you obtain when using the Trapezoidal vs. Simpson's rule, and with the exact value.

   iii. For each method (Trapezoidal and Simpson), how many slices do you need to approximate the integral with an error of $O(10^{-9})$? We are only looking for a rough estimate for the number of slices for each method. I.e., $N = 2^n$, with $n = 2,3,4\ldots$, 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?

   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.

   iv. Adapt the "practical estimation of errors" of the textbook (§ 5.2.1, p. 153) to the trapezoidal method *only* to obtain the error estimation for $N_2 = 32$ (using $N_1 = 16$).

   v. Why wouldn't the practical estimation method work with the Simpson's rule *in our particular case*? How (in a few words; no need to implement it) would we need to adapt the practical estimation of errors method for it to work with the Simpson's rule *in our particular case*?

   FOR THE WHOLE EXERCISE ON THIS INTEGRAL, SUBMIT YOUR CODE (IN A SEPARATE FILE FROM Q2(A)III), PRINTED OUTPUTS, AND WRITTEN ANSWERS.

   (b) Write a Python function that calculates the value of $J_n(x)$ from eqn. (5) using Simpson's rule with $N = 1000$ points. Use your function in a program to make a plot, on a single graph, of the Bessel functions J_0, J_3, and J_5 as a function of x from $x = 0$ to $x = 20$. Then make a second plot where you examine 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?

   HAND IN THE CODE, OUTPUT, PLOT(S), AND A BRIEF DESCRIPTION OF YOUR RESULTS.

   (c) Using the fact that $z_{m=3,n=2} \approx 11.620$ (we will learn methods to find zeros of functions later in the course), use your Simpson rule to create a three dimensional surface plot of $u_{m=3,n=2}$ from (4), for $0 \le r/R \le 1$, at $t = 0$. There are various ways to do this, one approach is to have a line like

   ```
   U[R>R0] = 0.0
   ```

which uses Numpy boolean masking to set all elements of the array `U` to zero where the condition `R>R0` is satisfied. Print various views of the surface plot to provide a good sense of what the function looks like.

HAND IN VARIOUS VIEWS OF THE SURFACE PLOTS ONLY.

3. **[20%] Electrostatic potential for a line of charge**

   (a) Referring to (8), write a code to calculate $V(r,z)$ using Simpson's rule. To make sure you are doing things correctly, you will need to compare your results to the known result in (9) using SciPy for the exact result and adjust the integration method until it falls within a desired accuracy.

   First, overlay your calculations of $V(r,z=0)$ using (9) and using Simpson's Rule to calculate (8) with `N = 8` over the range $0.25\,\text{mm} < r < 5\,\text{mm}$ in a plot. Set $Q = 10^{-13}\,\text{C}$ and $\ell = 1\,\text{mm}$. You can find the value of ϵ_0 online, or use the SciPy one. Even with `N = 8`, you should get a pretty good agreement between the two calculations.

   Then plot the difference of these quantities and see if you can bring down the fractional error to about one part in a million by increasing `N`.

   SUBMIT YOUR CODE, PLOTS, AND THE WRITTEN ANSWERS TO THE QUESTIONS.

   (b) With the value of `N` from Q3a, create a plot of the potential field in the domain $-5\,\text{mm} < z < 5\,\text{mm}$ and $0.25\,\text{mm} < r < 5\,\text{mm}$. Make sure to label the contours or otherwise indicate the value of V in volts. Also make sure to use enough resolution in r and z that you have captured the gradients related to the electric field correctly.

   SUBMIT YOUR CODE (YOU MAY COMBINE IT WITH THE PREVIOUS QUESTION'S CODE) AND PLOTS.