Computational Lab 2 : Numerical error, trapezoid and Simpson's rule

Due September 20th, 2019 (@ 17:00)

- Read this document and do its suggested readings to help with the pre-labs and labs.
- The topics 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.
- 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)
```

Physics Background

Electrostatic potential for a line of charge (for Q3): 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

$$\lambda(z) = \frac{Q}{l} e^{-z^2/l^2}, \tag{1}$$

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

$$V(r,z) = \int_{-\infty}^{\infty} \frac{\lambda(z')\,dz'}{4\pi\epsilon_0\sqrt{(z-z')^2+r^2}} = \int_{-\infty}^{\infty} \frac{Qe^{-z'^2/l^2}\,dz'}{4\pi\epsilon_0 l\sqrt{(z-z')^2+r^2}},\ r>0 \tag{2}$$

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 $z' = l\tan u$ and this results in:

$$V(r,z) = \int_{-\pi/2}^{\pi/2} \frac{Qe^{-(\tan u)^2}\,du}{4\pi\epsilon_0\cos^2 u\sqrt{(z-l\tan u)^2+r^2}}, \tag{3}$$

For $z=0$, a known result is that (2) becomes

$$V(r,z=0) = \frac{Q}{4\pi\epsilon_0 l}e^{r^2/2l^2}K_0(r^2/2l^2), \tag{4}$$

where $K_0(x)$ is a modified Bessel function (in Python it is implemented as `scipy.special.kn(0,x)`). Comparing your solution to this case will be useful to determine the accuracy of your integral calculation.

Computational Background

Reading data from a textfile The command

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

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

Numerical integrals The corresponding textbook sections are 5.1 – 5.3.

How to time the performance of your code. : See the first computational lab for notes on how to time your code.

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`,
- Error function: `scipy.special.erf`

Scipy constants Thanks to `scipy.constants`, you don't have to look up fundamental constants anymore! Follow the link below to see how:
https://docs.scipy.org/doc/scipy/reference/constants.html

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

$$\overline{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \sigma \equiv \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\overline{x})^2} \tag{5}$$

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,

$$\sigma \equiv \sqrt{\frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - n\overline{x}^2\right)}, \tag{6}$$

might seem preferable because, if you think about it, you can calculate σ in eqn. (6) 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. (5), and which you can think of as a correct calculation.

Numerical (roundoff) error: Read Section 4.2 of Newman's textbook, which discusses characteristics of machine error. One important point is that you can treat errors on numerical calculations as random and independent.[1] As a result of this, you can use standard error statistics as in experimental physics to figure out how error propagates through numerical calculations. This results in expressions like (4.7) in the text, describing the error on the sum (series) of N terms $x = x_1 + \ldots + x_N$:

$$\sigma = C\sqrt{N}\sqrt{\overline{x^2}}, \tag{7}$$

where $C \approx 10^{-16}$ is the *error constant* defined on p.130 and the overbar indicates a mean value. This means that the more numbers we include in a given series, the larger the error (by $O(\sqrt{N})$). Even if the mean error is small compared to the individual terms the *fractional error*

$$\frac{\sigma}{\sum_i x_i} = \frac{C}{\sqrt{N}}\frac{\sqrt{\overline{x^2}}}{\overline{x}} \tag{8}$$

can be really large if the mean value $\overline{x}$ is small, as is the case when we sum over large numbers of opposite sign.

Questions

1. **[30%] 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 methods (5) and (6), 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 (6) 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 (5) to calculate the standard deviation of Michelsen's speed of light data (in 10^3 km s^{-1}), which is stored in the file `cdata.txt`, and which was taken from John Baez's (U.C. Riverside) website `http://math.ucr.edu/home/baez/physics/Relativity/SpeedOfLight/measure_c.html`, back when it was still up.

   Calculate the relative error with respect to

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

   Now do the same for eqn. (6). Which relative error is larger in magnitude?

[1] This is a little confusing because you will find that errors on a given computer are often *reproducible*: they'll come out the same if you do the calculation the same way on your computer. But there is typically no way to predict what the error is — for the same operations it could be different on different computers, or even if you do a software update on your computer.

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 with

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

with the same standard deviation but a larger mean. Then evaluate the relative error of (5) and (6), 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.

SUBMIT PRINTED OUTPUT AND WRITTEN ANSWERS TO QUESTIONS.

2. **[25%] Exploring roundoff error** The idea of this exercise is to explore the effects of roundoff error for a polynomial calculated a couple of ways. Consider $p(u) = (1-u)^8$ in the vicinity of $u = 1$. Algebraically, this is equivalent to the following expansion

$$q(u) = 1 - 8u + 28u^2 - 56u^3 + 70u^4 - 56u^5 + 28u^6 - 8u^7 + u^8. \tag{9}$$

But numerically p and q are not exactly the same.

(a) the same graph, plot $p(u)$ and $q(u)$ very close to $u = 1$, for example picking 500 points in the range $0.98 < u < 1.02$. Which plot appears noisier? Can you explain why?

SUBMIT YOUR CODE AND PLOTS

(b) Now plot $p(u) - q(u)$ and the histogram of this quantity $p(u) - q(u)$ (for u near 1) using the `hist` function in `pylab`. Do you think there is a connection between the distribution in this histogram and the statistical quantity expressed in (7) above? To check, first calculate the standard deviation of this distribution (you can use the `std` function in `numpy`). Then calculate the estimate obtained by using equation (7), with $C = 10^{-16}$. State how you calculated the other terms in (7). *Hint: We are looking for order of magnitude consistency here, do not worry about $O(30-50\%)$ differences.*

SUBMIT YOUR WRITTEN ANSWERS TO THE QUESTIONS

(c) From equation (8) above show that for values of `u` somewhat greater than `0.980` but less than `1.0` the error is around 100%. Verify this by plotting or printing out `abs(p-q)/abs(p)` for `u` starting at `u = 0.980` and increasing slowly up to about `u = 0.984` (it might be different on your computer). This fractional error quantity is noisy and diverges quickly as `u` approaches `1.0` so you might need to plot or print several values to get a good estimate of the values of `u` at which the error approaches 100%.

SUBMIT PLOTS AND YOUR ANSWERS TO THE QUESTIONS

(d) Of course roundoff error doesn't just apply to series, it also comes up in products and quotients. For the same `u` near 1.0 as in Q1a-b, calculate the standard deviation (error) of the numerically calculated quantity `f = u**8/((u**4)*(u**4))`. This quantity will show a range of values around `1.0`, with roundoff error. You can get a sense of the error by plotting `f-1` versus `u`. Compare this error to the estimate in equation (4.5) on p.131 of the text (don't worry about the factor of $\sqrt{2}$).

SUBMIT YOUR WRITTEN ANSWERS TO THE QUESTIONS

3. **[15%] Stefan–Boltzmann constant** The black body function can be written as a function of wavenumber ν and temperature T as in equation 10, using the Planck constant h, the speed of light c and the Boltzmann constant k.

$$B = \frac{2hc^2\nu^3}{e^{\frac{hc\nu}{kT}} - 1} \tag{10}$$

(a) Show that the integration of B over ν can be written as

$$W = \pi \int_0^\infty B d\nu = C_1 \int_0^\infty \frac{x^3}{e^x - 1} dx. \tag{11}$$

What is in C_1?

NOTHING TO SUBMIT

(b) Write a program to calculate the value for B given the temperature T. Explain the method used to integrate over the infinite range, and give an estimate for the accuracy of the method.

SUBMIT YOUR CODE AND WRITTEN ANSWERS TO QUESTIONS.

(c) The total energy per unit area emitted by a black body follows Stefan's law (which you might have studied in the PHY224 "Black body" experiment). According to Stefan's law

$$W = \sigma T^4, \tag{12}$$

where σ is the Stefan–Boltzmann constant. Use your answer above to derive a value for the Stefan-Boltzmann constant in SI units to three significant figures or more. Check your result against the value given in `scipy.constants`.

SUBMIT YOUR CODE AND WRITTEN ANSWERS TO QUESTIONS.

4. **[30%] Electrostatic potential for a line of charge**

(a) Referring to (3), 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 (4) and adjust the integration method until it falls within a desired accuracy. The relevant scipy modified Bessel function call for (4) has the form `scipy.special.kn(0,x)`.

First, overlay your calculations of $V(r, z = 0)$ using (4) and using Simpson's Rule to calculate (3) with `N = 8` over the range $0.25\,\text{mm} < r < 5\,\text{mm}$ in a plot. Set $Q = 10^{-13}$ C and $l = 1$ mm. You can find the value of ϵ_0 online, in a book, or as follows:

```
from scipy.constants import epsilon_0
```

I found that even with `N = 8`, I got 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 part a, 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. 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 AND PLOTS.

Question:	1	2	3	4	Total
Points:	30	25	15	30	100