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

Nico Grisouard (Instructor), Alex Cabaj and Vince Pascuzzi (TAs),
Haruki Hirasawa (Marker)

Due Friday, 21 September 2018, 5 pm

General Advice

- 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.
- Carefully check what you are supposed to hand in for grading in the section "Lab Instructions".
- 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

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`.

Numerical integrals The corresponding textbook sections are 5.1 – 5.3.

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

Measuring execution time See the first lab for indications of a basic way to measure execution time.

Integrals over infinite ranges In class, and in the basic presentations of the trapezoidal and Simpson's rules for integration, we only treated finite ranges of integration:

$$\int_a^b f(x)\mathrm{d}x.$$

But what if one of or both bounds were replaced by an infinity? Section 5.8 (p. 179) of the textbook provides solutions.

Questions

1. **[35% of the lab] 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 (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 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. (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 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.

SUBMIT PRINTED OUTPUT AND WRITTEN ANSWERS TO QUESTIONS.

2. **[50% of the lab] Trapezoidal and Simpson's rules for integration and their errors**.

(a) We seek to evaluate the function

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2} \mathrm{d}t \tag{3}$$

at $x = 3$.

i. For $N = 10$ slices, compare the value you obtain when using the Trapezoidal vs. Simpson's rule, and with the value of `scipy`'s `erf` function.

ii. For each method (Trapezoidal and Simpson), how many slices do you need to approximate the integral with an error of $O(10^{-11})$? Use `scipy`'s value as the "true" value. We are only looking for an order of magnitude for the number of slices for each method. I.e., $N = 10^n$, with $n \in \mathbb{N}$, 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 a relative error of $O(10^{-11})$ 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.

iii. Adapt the "practical estimation of errors" of the textbook (§ 5.2.1, p. 153) to both methods to obtain an error estimation. Like in i., start with $N_1 = 10$.

iv. This case is simple enough (i.e., all functions are analytical) that you can compare those results with the Euler-Maclaurin formulas. Do it and comment. In particular, why don't the two methods give the exact same result?

Note: be careful, not to compare apples and oranges. In order to compare the Euler-Maclaurin and practical estimates, you have to choose the correct integration step!

Other note: there is a typo in eqn. (5.24) of the textbook: 90 should be 180.

FOR EVERY SUB-QUESTION, SUBMIT YOUR CODE, PRINTED OUTPUT, AND WRITTEN ANSWERS TO QUESTIONS.

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

- Do Exercise 5.4a,

- 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 5.4b. 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.

SUBMIT YOUR CODE, PLOTS, PRINTED OUTPUTS, AND WRITTEN ANSWERS.

3. **[15% of the lab] Stefan-Boltzmann constant.** (Exercise 5.12, p.181).

5.12A: NOTHING TO SUBMIT

(a) Do 5.12b. *Note: python will not like divisions by zero, even if the mathematical limits of the integrands are finite. Therefore, after the change of variable presented in the computational background, you may have to integrate from* δ *to* $1-\delta$*, with* $0 < \delta \ll 1$*, rather than from 0 to 1.*

SUBMIT YOUR CODE AND WRITTEN ANSWERS TO QUESTIONS.

(b) Do 5.12c, and compare your result with the value given by `scipy.constants`.

SUBMIT PRINTED OUTPUT, AND WRITTEN ANSWERS TO QUESTION.