Lab assignment #2: Numerical Errors and Integration

Due Friday, September 23 2022, 5 pm

General Advice

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..
- Test your code as you go, **not** just 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.
- Recall, one way to practice modularity is to define external functions for repetitive tasks. For example, you may want to write generic functions for Trapezoidal and Simpson's rules (or use the piece of code provided by the textbook online resources), place them in a separate file, and call and use them in your answer files.

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.

Reading data from a textfile The command

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

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

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

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

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.

The canned routine `numpy.std(array, ddof=1)` also implements eqn. (1). 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. (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 multiple times on your computer. But there is typically no way to predict what this error value is, i.e. it could be different on different computers, or even if you do a software update on your computer.) 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 $x = x_1 + \ldots + x_N$:

$$\sigma = C\sqrt{N}\sqrt{\overline{x^2}}, \tag{3}$$

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*

$$\frac{\sigma}{\sum_i x_i} = \frac{C}{\sqrt{N}}\frac{\sqrt{\overline{x^2}}}{\overline{x}} \tag{4}$$

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.

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`

Physics Background

Black body radiation The black body function can be written as a function of wavenumber ν and temperature T, using the Planck constant h, the speed of light c and the Boltzmann constant k:

$$B = \frac{2hc^2\nu^3}{e^{\frac{hc\nu}{kT}} - 1} \tag{5}$$

The total energy per unit area emitted by a black body follows Stefan's law:

$$W = \sigma T^4, \tag{6}$$

where σ is the Stefan-Boltzmann constant. (Of course, you remember all this if I was your prof for PHY252.)

Questions

1. **[25%] 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. (Stop and think about why this check is 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: `http://math.ucr.edu/home/baez/physics/Relativity/SpeedOfLight/measure_c.html`

   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 answer.

(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 answer.

(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 pseudo-code, code, printed output and written answer.

2. **[25%] Trapezoidal and Simpson's rules for integration**

We seek to evaluate the integral

$$\int_0^1 \frac{4}{1+x^2} dx. \tag{7}$$

(a) What is the exact value of this integral?

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

(c) 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.

(d) 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 $N_1 = 16$).

(e) 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 pseudo-code, code, printed outputs, and written answers.

3. **[25%] Stefan-Boltzmann constant**

(a) Show that in eqn (5), the integration of B over ν can be written as

$$W = \pi \int_0^\infty B d\nu = C_1 \int_0^\infty \frac{x^3}{e^x - 1} dx. \tag{8}$$

What is in C_1?

Submit your written answer.

(b) Write a program to calculate the value for W 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 pseudocode, code, and written answer.

(c) Use your answer above to derive a value for the Stefan-Boltzmann constant (see eqn 6) in SI units, to three significant figures or more. Check your result against the value given in `scipy.constants`.

Submit your code, and written answer.

4. **[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) On 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 graph

(b) Now plot $p(u) - q(u)$, and the histogram of this quantity $p(u) - q(u)$ (for u near 1). Do you think there is a connection between the distribution in this histogram and the statistical quantity expressed in eqn (3) 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 (3), with $C = 10^{-16}$.

State how you calculated the other terms in (3). *Hint: We are looking for order of magnitude consistency here, do not worry about $O(30-50\%)$ differences.*

Submit your plots and written answer

(c) From equation (4) 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 different computers). 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 your plots and written answer

(d) 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 parts a-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 textbook (don't worry about the factor of $\sqrt{2}$).

Submit your plot(s) and written answer