Lab Report 1: Numerical Errors and Simple Integration

PHY407 Fall 2023

Background

Solving integrals numerically: Lecture notes, as well as Sections 5.1 through 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.

Numerical (roundoff) error: Read Section 4.2 of the 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 $x = x_1 + \ldots + x_N$:

$$\sigma = C\sqrt{N}\sqrt{\overline{x^2}}, \tag{1}$$

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*

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

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.

Diffraction of light in telescopes: *(From page 148 of the textbook)* Our ability to resolve detail in astronomical observations is limited by the diffraction of light in our telescopes. Light from stars can be treated effectively as coming from a point source at infinity. When such light, with wavelength λ, passes through the circular aperture of a telescope and is focused by the telescope, it doesn't produce a single dot in the focal plane. Rather, it produces a circular diffraction pattern, consisting of a central spot surrounded by a series of concentric rings. The intensity I of the light in this diffraction pattern, assuming the telescope aperture has unit radius, is given by:

$$I(r) = \left(\frac{J_1(kr)}{kr}\right)^2 \tag{3}$$

where r is the distance in the focal plane from the center of the diffraction pattern, $k = 2\pi/\lambda$, and $J_1(x)$ is a Bessel function.

Bessel functions: These appear in various physics problems, and are given by:

$$J_m(x) = \frac{1}{\pi}\int_0^\pi \cos(m\theta - x\sin\theta)d\theta$$

where m is a whole number and x is a non-negative real number. Hint for this lab:

$$\lim_{x\to 0}\left(\frac{J_1(x)}{x}\right) = 1/2$$

SciPy special functions: These are a little too exotic to be part of NumPy, but common enough to have pre-coded versions in SciPy. Import them at the beginning of your code: `from scipy.special import XX` (with `XX` the function you want to use, for example `jv` for Bessel functions and `dawsn` for Dawson's integral.)

Density plots: When using `matplotlib.pyplot` for density plots, you may find the default colour scheme doesn't work very well, e.g. some features in the density plot are too difficult to see. A simple way to deal with this problem is to use one of the other colour schemes (e.g. "hot") for density plots described in Section 3.3 of the textbook. For a more sophisticated solution, the `imshow` function has an additional optional argument `vmax` that allows you to set the value that corresponds to the brightest point in the plot. For example, `imshow(x,vmax=0.1)` makes elements in x with value 0.1 or greater produce the brightest (most positive) colour. By lowering the `vmax` value, you can reduce the total range of values between the minimum and maximum brightness, hence increasing the sensitivity of the plot and making subtle details more visible. Similarly, there is also an optional `vmin` argument that can be used to set the value that corresponds to the dimmest (most negative) colour.

Optionally, you can also try `matplotlib.pyplot.pcolormesh` as an alternative to `imshow`.

Questions

Note, each of the following 3 questions is equally weighted in the marking scheme.

1. **Diffraction**

   (a) Write your own Python function that calculates the value of the Bessel function $J_m(x)$, taking m and x as inputs, using Simpson's rule with $N = 1000$ points. Use this function in a program to make a plot, on a single graph, of the Bessel functions J_0, J_1, J_2 as a function of x from $x = 0$ to $x = 20$.

   (b) Compare graphically the difference between the results of your Bessel function and those from `scipy.special.jv`.

   (c) Make a program that creates a density plot of the intensity of the circular diffraction pattern of a point light source with $\lambda = 500$nm, in a square region of the focal plane of a telescope. (See equation 3.) The plot should cover values of r from 0 up to about 1μm. *Hint: if you are using* `imshow`, *try setting* `vmax = 0.01`.

   For this entire question, submit your pseudo-code, code, and plots.

2. **Trapezoidal and Simpson's rules for integration**

   We seek to evaluate the Dawson function at $x = 4$:

$$D(x) = \exp -x^2 \int_0^x \exp t^2 dt. \tag{4}$$

   (a) 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`.

   (b) For each method (Trapezoidal and Simpson): how many slices do you need to approximate the integral with an error of $O(10^{-9})$, and how long does it takes to compute the integral with this required level of accuracy? Note:

   - You should treat the SciPy 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 $n = 2, 3, 4 \ldots$, and N or n being the answer).
   - Find the required number of slices using the simplest method: keep increasing the number of slices until you reach the required accuracy.
   - The integration is quite fast, so to get an accurate timing, you may want to repeat the same integration many times over.
   - Recall that we described a timing method in the lab exercises. You can use this method, or optionally, you can try more accurate methods.

   (c) Adapt the "practical estimation of errors" of the textbook (Section 5.2.1, page 153) to both methods to obtain the error estimate for $N_2 = 64$ (using $N_1 = 32$).

   For this entire question, submit your code, printed outputs, and written answers.

3. **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:

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

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 equation (1)? 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 (1), with $C = 10^{-16}$. State how you calculated the other terms in (1). *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 (2) 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 page 131 of the textbook (don't worry about the factor of $\sqrt{2}$).

Submit your plot(s) and written answer