HW #1 Basics

Answer the following questions. Hand in your (documented) code along with your solutions. You may want to create a github repository with your code, which will make keeping track of changes and submitting your code easier.

1. **Floating away ...**

In this problem you will get to know your computer a little bit better.

a) (Z ex1.1) In your choice of programming language, create a floating point variable and initialize it to 0.1. Now, print it out in full precision (you may need to use format specifiers in your language to get all significant digits the computer tracks).

You should see that it is not exactly 0.1 to the computer—this is the floating point error. The number 0.1 is not exactly representable in the binary format used for floating point. What is the degree of floating point error you find? How does this floating point error change if you declare variable as "single precision" versus "double precision"?

b) (Z ex1.2) Using the method you sketched out in class, determine the roundoff error, ϵ, for your machine.

c) For kicks, check in with a couple of your classmates who have different hardware and/or operating systems. Compare your answers for (a) and (b). What do you find?

2. **Integral Processes.**

Numerical integration is important for solving ordinary differential equations, among other data analysis tasks. Previously you've likely used a built-in function in Python or the equivalent in another programming language. You may have treated it as a black-box and didn't think too closely about it's degree of accuracy. In this problem, you're going to dig into the integration black-box a little bit more. First, you'll start by writing you own integration routines to evaluate a general integral:

$$I = \int_a^b f(x)dx. \tag{1}$$

I. One of the simplest methods for integration as we discussed in class uses the Rectangle Rule:

$$I = \sum_{i=1}^{N} f(x_i)\Delta x, \tag{2}$$

where N is the number of (equal-sized) integration steps and $\Delta x = (b-a)/N$, $x_i = a + (i-1)\Delta x$.

II. Another *slightly* more sophisticated approach uses the Trapezoid Rule:

$$I = \Delta x \sum_{i=1}^{N} \frac{f(x_i) + f(x_{i+1})}{2}. \tag{3}$$

This may seem to require twice as many function evaluations as the method above, but there is a simple way to avoid this, which you should implement.

a) Numerically compute the integral

$$\int_1^5 \frac{1}{x^{3/2}} dx \tag{4}$$

with both methods above, and plot the error in the numerical integral against the step size Δx for both methods. Approximately how many steps are required to get an answer with a fractional error $|I - I_{exact}|/I_{exact} < 10^{-3}$ for the rectangle and the trapezoid rule? What about 10^{-5}?

What did you lean about the trade-off between method, accuracy and calculation speed?[1]

b) Compare the results of your two integration routines to a built-in function in your programming language (or another common package like Mathematica or Matlab). For example, if you're using Python, you may use one of the functions available in the *scipy.integrate* module. In Mathematica, you could use the *Integrate* function. What is the order accuracy of the black-box method? What is the default approach to integration? What step sizes do you need above to obtain a similar result?

3. **Shooting for the stars.**

Now let's use our integration routines to evaluate an astronomy problem! Choose *one* of the following and use one of the integration methods you coded above to solve it:

a) Write a program to compute the orbit of the Earth around the Sun (follow Z ex 1.8). This program can be used to compute the orbits of different solar system bodies or modified to be applied to other stellar systems.

b) Write a program that computes the integral

$$D_c = \int_0^z dz' [\Omega_m (1+z')^3 + (1. - \Omega_m - \Omega_\Lambda)(1+z')^2 + \Omega_\Lambda]^{-1} \tag{5}$$

given input values of Ω_m, Ω_Λ and z. If you look up Hogg (1999, astro-ph/99055116), equation (15) you will see that this integral, multiplied by $cH_0^{-1} = 3000h^{-1}$ Mpc, gives the "comoving distance" to an object at redshift z in a universe with matter density parameter Ω_m and

[1]No timing necessary. The number of floating point operations is a good proxy for the evaluation time (assuming an otherwise efficient program

cosmological constant Ω_Λ. This can in turn be used to calculate (with no additional integrals) other cosmologically useful distances like the luminosity or angular diameter distance.

What is the comoving distance to $z = 2$ in a universe with $\Omega_m = 0.3$ and $\Omega_\Lambda = 0.7$? Make a plot of the comoving distance in Mpc versus redshift for $z = 0 - 10$.