Physics 307 Homework 2
Due Thursday, 19 September

This week, you will extend and modify your integration routine to study the numerical error in integration.

Remember: this class intends for you to ask us lots of questions as you work. Computational physics is best learned in small conversations in the context of you trying to solve problems, not from a lecture. Ask us stuff early and often:

- in person in class
- over Slack
- over email

1. Post your favorite meme to the #physics307 channel on Slack. (This is mandatory and we will not accept the assignment until you do this. The point is to make sure everyone is on the Slack channel!)
2. Write a separate function that performs your integration. It should take as arguments:
   - The lower bound of the integration interval
   - The upper bound of the integration interval
   - The number of integration bins to use. (You could pass in the stepsize instead, but in that case make sure you only use stepsizes that go into the interval evenly!)

   This will make your life much easier in the long run.
3. Analytically compute the value of

$$\int_0^2 \sin x \, dx.$$

   Then use your integration routine to compute it numerically, using a range of Riemann sum stepsizes from 10^0 to 10^{-7}. It will be easiest if you automate this last process by using a `for` loop to cycle through the various values of your stepsize; this is where putting your integrator in a separate function will save you work! Do this using the left-hand rule, if you have been doing something else. Then make a log-log plot of the error (magnitude of the difference between the analytic and numeric answers) vs. the stepsize.

   Ensure that the interval (0,2) can be divided evenly into intervals of the stepsize you choose! (What happens if this isn't the case?)

4. Now, write another integration routine that uses the midpoint rule. (Copy-paste from your previous function may save you a lot of work...) Again make a log-log plot of the error as a function of the stepsize, using stepsizes from 10^0 to 10^{-7}. Plot these graphs for both the midpoint rule and the left-hand rule on the same axis. Do the graphs look like you expect? How do you explain their behavior for large stepsizes? For very small stepsizes? How do you interpret their slopes?

5. Repeat the previous problem using the trapezoid rule. Does it behave as you expect it to?

6. Modify any or all of your programs to use `double` variables rather than `float`s, if you are working in C. Does this have the effect you expect on the error?

7. Finally, use one of your second-order integrators to compute

$$\int_0^2 \sqrt{x}\, dx.$$

   Plot the error vs. the stepsize again. Why do you think this is behaving badly? Hint: Compare this to a similar plot for

$$\int_1^2 \sqrt{x}\, dx.$$

8. Extra credit: Write an integrator that uses Simpson's rule. As we will discuss in class, the easiest way to do the Simpson's rule computation is to take 1/3 of the trapezoid rule result and add it to 2/3 of the midpoint rule result. Make the log-log error plot as before. Does it perform as you expect?

FAQ:

- *Some of my data don't show! I get errors when I try to plot!*
  - Look at your data file and make sure your values are sensible. Remember that if you're making a log-log plot, you can only plot positive values – you may need the `fabs()` function (C) or the `abs()` function (Python) to take absolute values.
- *I get funny data! My error goes down for smaller stepsizes like I expect, but then back up again!*
  - Did you finish the fifth part, where you use `double` variables? What happens then? Does it behave like you expect now? Does this give you any insight into what is going on?

- *What stepsizes should I use?*

  - You want a range that's evenly scattered – in *logarithmic space* – in the range. So, instead of *adding* a fixed value to your stepsize in your outer `for` loop, maybe you could *multiply* (or divide) by a fixed value? In Python, either write a `while` loop to do the integration, or investigate the `logspace` function that is part of `numpy`.

- *My error bounces around strangely!*

  - Remember the "last step bug" that we might talk about in class. In brief, you want `for(x=a; x<b-dx/2; x=x+dx)` to avoid the rounding issue that happens at the last step. It is even better to do `for (bin=0; bin<nbins; bin=bin+1)` and then calculate values of `x` on the fly, to avoid rounding issues entirely.

- *My integrator is stuck – it is suddenly taking forever to run and not stopping!*

  - Ask Walter about this, especially if you are using C. There's a subtle but interesting rounding bug that can happen.